Pós-Graduação em Ciência da Computação

EVANDRO JOSÉ DA ROCHA E SILVA

Geração Dinâmica de Protótipos para Classificação em Bases de Dados com Múltiplas Classes Desbalanceadas

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

EVANDRO JOSÉ DA ROCHA E SILVA

Geração Dinâmica de Protótipos para Classificação em Bases de Dados com Múltiplas Classes Desbalanceadas

Tese apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciências da Computação.

Área de Concentração: Inteligência Computacional e Reconhecimento de Padrões.

Orientador: Prof. Dr. Cleber Zanchettin

Recife

2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2105

S586g Silva, Evandro José da Rocha e
Geração Dinâmica de Protótipos para Classificação em Bases de Dados com Múltiplas Classes Desbalanceadas/ Evandro José da Rocha e Silva – 2019.
192 f., fig., tab.

Orientador: Cleber Zanchettin.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação. Recife, 2019.
Inclui referências e apêndices.

1. Inteligência Computacional 2. Aprendizado de Máquina. 3. Bases Desbalanceadas. 4. Pré-processamento de Dados. I. Zanchettin, Cleber (orientador). II. Título.

006.31 CDD (22. ed.) UFPE-CCEN 2020-02

Evandro José da Rocha e Silva

“Geração Dinâmica de Protótipos para Classificação em Bases de Dados com Múltiplas Classes Desbalanceadas”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Aprovado em: 06/09/2019.

_____

Orientador: Prof. Dr. Cleber Zanchettin

BANCA EXAMINADORA

_____

Profa. Dra.Teresa Bernarda Ludermir
Centro de Informática/UFPE

_____

Prof. Dr. Leandro Maciel Almeida
Centro de Informática / UFPE

_____

Prof. Dr. Ricardo de Andrade Lira Rabêlo
Departamento de Computação/UFPI

_____

Prof. Dr. Byron Leite Dantas Bezerra
Escola Politécnica de Pernambuco/UPE

_____

Prof. Dr. João Fausto Lorenzato de Oliveira
Escola Politécnica de Pernambuco/UPE

A meu pai e minha mãe, a quem muito amo.

AGRADECIMENTOS

Agradeço primeiramente a Deus. A Ele toda honra, glória e louvor, para sempre e sempre.

Agradeço também aos meus pais, familiares e amigos pelo suporte durante esta longa jornada.

Por fim, agradeço ao meu orientador, e todos os professores e colegas pelos ensinamentos e desafios que me permitiram chegar até este ponto.

RESUMO

Algoritmos de Aprendizado de Máquina comumente assumem que no conjunto de treinamento do modelo a quantidade de observações de cada classe é igual ou bastante similar. Entretanto, muitas bases de dados possuem classes com quantidades significativamente diferentes de instâncias. Foi observado na literatura que tais diferenças provocam uma queda no desempenho dos classificadores, a qual é associada ao enviesamento causado pela influência das classes que possuem mais instâncias. O efeito negativo no desempenho de classificadores é associado também à sobreposição de bordas, pequenos disjuntos, classes raras ou extremamente raras, e *data set shift*. Existem vários estudos considerando o desbalanceamento em bases de dados com duas classes, porém, estudos com múltiplas classes são menos frequentes, normalmente associados a sua maior complexidade. As soluções existentes normalmente englobam a decomposição do problema em combinações de versões binárias ou propostas *ad hoc*, ou seja, soluções específicas para o problema. Esta tese apresenta um estudo sobre a utilização de Geração de Protótipos no problema de múltiplas classes desbalanceadas. Duas abordagens foram propostas para tratar o problema, VDBC (*Voronoi Diagram Based Classifier*) e DCIA (*Dynamic Centroid Insertion and Adjustment*). A primeira consiste na geração de protótipos a partir da análise da vizinhança de cada instância na base de dados. A segunda aborda a geração mínima de protótipos, os quais podem ter seu posicionamento ajustado para refletir melhor as regiões de representação das classes. A partir das abordagens propostas, foram investigadas variações dos modelos, as quais foram comparadas entre si. Foi possível observar que o DCIA se destaca em relação ao VDBC. As suas melhores variações, cujos desempenhos são estatisticamente equivalentes, foram comparadas com várias soluções encontradas na literatura. Os resultados obtidos demostram a eficácia do DCIA ao ser competitivo e obter o melhor desempenho em várias das bases de dados utilizadas na validação das abordagens, principalmente nas bases que são consideradas mais desbalanceadas.

Palavras-chaves: Aprendizado de Máquina. Bases Desbalanceadas. Geração de Protótipos. Múltiplas Classes Desbalanceadas. Pré-processamento de Dados.

ABSTRACT

Machine Learning algorithms usually assume that in the model's training data, the amount of observations for each class is equal or quite similar. However, several data sets have classes with significantly different number of instances. In the literature, it was observed that such differences induce classifiers to lose performance, which is associated with the bias caused by the influence of classes that have more instances. The negative effect on classifiers performance is also associated with border overlapping, small disjuncts, rare or extremely rare classes, and data set shift. There are several studies considering imbalance in data sets with two classes. However, studies with multiple classes are less frequent, usually associated with greater complexity. Existing solutions typically involve a problem decomposition into combinations of binary versions or *ad hoc* proposals, i.e., specific solutions for the problem. This thesis presents a study on the use of Prototype Generation with the problem of multiple imbalanced classes. Two approaches were proposed to deal with the problem, Voronoi Diagram Based Classifier (VDBC) and Dynamic Centroid Insertion and Adjustment (DCIA). The first one consists in the generation of prototypes from the neighborhood analysis of each instance in the data set. The second approach deals with a minimal prototype generation, which can have their positioning adjusted to better reflect the representation regions of classes. From the proposed approaches, variations of the models were investigated, which were compared with each other. It was possible to observe that DCIA stands out in relation to VDBC. DCIA's best variations, whose performances are statistically equivalent, were compared with various solutions found in the literature. The obtained results demonstrate the effectiveness of DCIA by achieving the best performance on several of the used data sets, mainly in the data sets that are considered more imbalanced.

Keywords: Machine Learning. Imbalanced Data Sets. Prototype Generation. Multiple Imbalanced Classes. Data Preprocessing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADASYN	*Adaptive Synthetic sampling approach*
AG	Algoritmo Genético
AM	Aprendizado de Máquina
AMCS	*Adaptive Multiple Classifier System*
AMDO	*Adaptive Mahalanobis Distance-based Over-sampling*
ASGP	*Adaptive Self-Generating Prototype*
AUC	*Area Under ROC Curve*
CBA	*Class Balance Accuracy*
CD	*Critical Difference*
CDD	*Critical Difference Diagram*
CNN	*Convolutional Neural Networks*
CoMBo	*Confusion Matrix Boosting*
DCIA	*Dynamic Centroid Insertion and Adjustment*
DCIACSO	*DCIA with Competitive Swarm Optimizer*
DCIAGSA	*DCIA with Gravity Search Algorithm*
DCIAPSO	*DCIA with Particle Swarm Optimization*
DECOC	*Diversified Error Correcting Output Codes*
DNN	*Deep Neural Networks*
DOVO	*Diversified One-Versus-One*
DSRBF	*Dynamic SMOTE RBF*
EASGP	*Evolutionary Adaptive Self-Generating Prototype*
ECOC	*Error Correcting Output Codes*
ELM	*Extreme Learning Machine*
FRIPS	*Fuzzy Rough Imbalanced Prototype Selection*
FSCNCA	*Feature Selection for Classification Using Neighborhood Component Analysis*
GP	Geração de Protótipos
GSA	*Gravity Search Algorithm*
GSVM-RU	*Granular SVMs-Repetitive Undersampling*
HDDT	*Hellinger Distance Decision Tree*

IBA	*[Generalized] Index of Balanced Accuracy*
ID	*Imbalance Degree*
IPADE-ID	*Iterative Prototype Adjustment Based on Differential Evolution for Imbalanced Domains*
IR	*Imbalance Ratio*
k-NN	*k-Nearest Neighbors*
MAvA	*Macro Average Arithmetic*
MAvG	*Macro Average Geometric*
MDO	*Mahalanobis Distance-based Over-sampling*
MIR	*Multiclass Imbalance Ratio*
MLP	*Multilayer Perceptron*
MRBF	*Memetic Radial Basis Function*
MROS	ROS adaptado para múltiplas classes
MRUS	RUS adaptado para múltiplas classes
MS	Mínima Sensibilidade
NCL	*Neigborhood Cleaning rule*
OAA	*One-Against-All*
OAO	*One-Against-One*
OP	*Optimized Precision*
OSBE	*Over-Sampling Balanced Ensemble*
OSS	*One-Sided Sampling*
OVA	*One-versus-All*
OVO	*One-versus-One*
RBF	*Radial Basis Function*
RNP	Redes Neurais Profundas
ROC	*Receiver Operating Characteristics*
ROS	*Random Oversampling*
RUS	*Random Undersampling*
SA	Seleção de Atributos
SBC	*[under]Sampling Based on Clustering*
SGSA	*Simple Gravity Search Algorithm*
SMOTE	*Synthetic Minority Oversampling Technique*
SP	Seleção de Protótipos

SPIDER	*Selective Preprocessing of Imbalanced Data*
SSRBF	*Static SMOTE RBF*
SVDD	*Support Vector Data Description*
SVM	*Support Vector Machine*
t-SNE	*t-distributed Stochastic Neighbor Embedding*
TVN	Taxa de Verdadeiro Negativo
TVP	Taxa de Verdeiro Positivo
USBE	*Under-Sampling Balanced Ensemble*
UVT	Um-versus-Todos
UVU	Um-versus-Um
VDBC	*Voronoi Diagram Based Classifier*
WELM	*Weighted ELM*
WOA-CM	*Whale Optimization Algorithm with Crossover and Mutation operators*

LISTA DE SÍMBOLOS

α	Alfa
$\mathcal{H}$	Arquivo de valores de *fitness*, ou aptidão
β	Beta
$\Delta \mid \delta$	Delta
ϵ	Epsilon
$\equiv$	Equivalente
γ	Gama
ι	Iota
λ	Lambda
$\mathcal{M}$	Métrica
$\cdot$	Multiplicação, elemento por elemento
$\in$	Pertence
ϕ	Phi
π	Pi
$\prod$	Produtório
σ	Sigma
$\sum$	Somatório
ζ	Zeta

SUMÁRIO

1 INTRODUÇÃO

Desenvolver algoritmos que permitam capacitar a máquina ao aprendizado após um processo de observação faz parte dos objetivos da área de Aprendizado de Máquina (AM). Tais algoritmos procuram examinar um conjunto de dados para prover informações que permitam a máquina absorver e generalizar o conhecimento presente nos dados.

Uma das maneiras de se gerar informação a partir de um conjunto de dados se dá com a utilização de técnicas de detecção de padrões. Assume-se que elementos de uma mesma classe possuem atributos com distribuições semelhantes. Os algoritmos de aprendizado procuram, então, identificar os padrões que podem ser observados para cada classe ou agrupamento.

Os conjuntos de dados podem possuir rótulos associados aos dados de entrada ou não. Quando os algoritmos têm acesso aos rótulos de um conjunto de dados para explorar a relação entre a entrada e a saída, os mesmos são ditos algoritmos de aprendizado supervisionado. Por outro lado, na ausência dos rótulos, os algoritmos consideram a similaridade entre as informações de entrada para realizar o mapeamento ou algum tipo de correlação entre eles. Neste caso passam a ser conhecidos como algoritmos de aprendizado não supervisionado. Nesta tese de doutorado é investigado a utilização de algoritmos de aprendizado supervisionado sobre um dos problemas inerentes a este tipo de aprendizagem de máquina.

A qualidade dos dados disponíveis relacionados ao problema investigado é essencial no desempenho dos modelos de aprendizagem de máquina. A representatividade, distribuição, e quantidade de exemplos ou instâncias são grandezas que influenciam o desempenho destes modelos. Muitas vezes, as amostras disponíveis em cada uma das classes estão balanceadas, ou seja, a quantidade de exemplos (instâncias) de cada classe é igual, ou muito parecida nas várias classes existentes na base de dados. Por outro lado, existem casos cujas classes se encontram desbalanceadas. Nestas bases, uma ou mais classes possuem uma diferença significativa quanto a quantidade de instâncias disponíveis para o treinamento dos modelos.

Quando expostos a bases desbalanceadas, os classificadores costumam sofrer queda no desempenho (normalmente associado ao erro de classificação do modelo) (BATISTA, 2003; NGUYEN; BOUZERDOUM; PHUNG, 2009). Além disso, métricas que são comumente usadas em AM podem ser enganosas, visto que podem apresentar um desempenho muito melhor nas simulações realizadas em laboratório do que em aplicações em campo (BRODERSEN et al., 2010; HU; DONG, 2014; WANG, 2014; LÓPEZ et al., 2013; PRATI; BATISTA; MONARD, 2004; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). Ademais, se descobriu que o problema, com sua consequente queda de desempenho, não advém somente da diferença na quantidade de instâncias entre as classes e do uso de métricas inapropriadas. Existe também relação

com as características próprias dos classificadores (JAPKOWICZ; STEPHEN, 2002), complexidade dos dados (JAPKOWICZ; STEPHEN, 2002; NGUYEN; BOUZERDOUM; PHUNG, 2009), *data set shift* (LÓPEZ et al., 2013), pequenos disjuntos (JAPKOWICZ; STEPHEN, 2002; NAPIERALA; STEFANOWSKI, 2012; LÓPEZ et al., 2013), presença de classes raras (NAPIERALA; STEFANOWSKI, 2012), ruído (NGUYEN; BOUZERDOUM; PHUNG, 2009; NAPIERALA; STEFANOWSKI, 2012; LÓPEZ et al., 2013), sobreposição de bordas entre as classes (PRATI; BATISTA; MONARD, 2004; NGUYEN; BOUZERDOUM; PHUNG, 2009; NAPIERALA; STEFANOWSKI, 2012; LÓPEZ et al., 2013; MALDONADO; MONTECINOS, 2014) e tamanho do conjunto de treinamento[1] (JAPKOWICZ; STEPHEN, 2002; NGUYEN; BOUZERDOUM; PHUNG, 2009; LÓPEZ et al., 2013). Além de ser bastante comum (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011), este problema é considerado como bastante desafiador na área (WANG, 2014).

A partir da década de 2000 o interesse sobre o tema vem crescendo constantemente (CHAWLA; JAPKOWICZ; KOTCZ, 2004; HAIXIANG et al., 2017), resultando em vários estudos e propostas para lidar com bases de dados desbalanceadas. Desses trabalhos, três tópicos principais podem ser destacados (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012): (1) formas de se abordar o problema (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LÓPEZ et al., 2013; NGUYEN; BOUZERDOUM; PHUNG, 2009; WANG, 2014; MALDONADO; MONTECINOS, 2014); (2) estudo sobre métricas (BRODERSEN et al., 2010; GARCÍA; MOLLINEDA; SANCHEZ, 2010; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012) e (3) estudo sobre o problema (JAPKOWICZ; STEPHEN, 2002; PRATI; BATISTA; MONARD, 2004; NGUYEN; BOUZERDOUM; PHUNG, 2009; LÓPEZ et al., 2013; NAPIERALA; STEFANOWSKI, 2012; MALDONADO; MONTECINOS, 2014; SILVA; ZANCHETTIN, 2015; SILVA; ZANCHETTIN, 2016a).

O primeiro tópico reúne os algoritmos criados ou adaptados para lidar com o problema de desbalanceamento nos dados. O segundo tópico estuda quais métricas são mais apropriadas para avaliar modelos de classificação sobre este tipo de base de dados. O último tópico se refere aos estudos que procuram entender o problema como um todo. Apesar dos três tópicos serem facilmente definidos, os mesmos são interdependentes.

Dentre os três tópicos, o primeiro é um dos mais investigados, normalmente associado a duas abordagens, conhecidas como abordagem em nível de dados e abordagem em nível de algoritmo (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). A primeira abordagem, em geral, procura pré-processar um conjunto de treinamento em busca de classes balanceadas. A segunda abordagem foca em adaptar classificadores ou modelos de classificação para lidar com o desbalanceamento dos dados (LÓPEZ et al., 2013).

Os estudos que abordam o problema de classes desbalanceadas foram bastante centrados em bases de dados binárias (FERNÁNDEZ et al., 2013; WANG; YAO, 2012; SILVA; ZANCHETTIN, 2016a; KOÇO; CAPPONI, 2013). A menor complexidade para se lidar com apenas duas classes foi um fator que contribuiu muito para isso (GALAR et al., 2011;

[1] Todos os termos técnicos citados são devidamente apresentados e explicados no Capítulo 2.

FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011). Entretanto, existem diversas bases desbalanceadas que possuem três ou mais classes, sendo possivelmente o caso mais comum.

Na literatura existem comumente três formas de se lidar com múltiplas classes desbalanceadas. Duas dessas formas utilizam técnicas de decomposição (ou binarização), transformando o problema de múltiplas classes em múltiplos problemas de duas classes (FERNÁNDEZ et al., 2013; HASTIE; TIBSHIRANI, 1998; GALAR et al., 2011; WANG; YAO, 2012). Tais técnicas se tornaram as mais comuns. Outra vez, a menor complexidade de se lidar com apenas duas classes foi um fator decisivo (GALAR et al., 2011; FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011). Neste caso, as classes do problema são separadas em pares e se treina um classificador para cada par. Ou ainda, as classes são separadas individualmente, e se treina um classificador para cada uma delas.

A terceira forma de se lidar com o problema é através de uma abordagem direta, ou seja, considerando todas as classes ao mesmo tempo. Apesar dessa forma *a priori* ser mais elaborada que uma técnica de binarização, o processo inteiro pode ser menos complexo. Por exemplo, ao se dividir uma base em vários pares de classes, será exigido menos de cada classificador, uma vez que cada um só terá de aprender a distribuição estatística entre duas classes. O modelo final, entretanto, deverá lidar com as respostas de todos os classificadores e combiná-las de forma a produzir o melhor desempenho possível. A maneira de combinar tais saídas pode ainda se tornar um desafio (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011; OU; MURPHEY, 2007; GALAR et al., 2011). Uma abordagem direta, por outro lado, além de permitir um modelo final menos complexo, pode também resultar em um melhor desempenho de classificação (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011; OU; MURPHEY, 2007).

1.1 HIPÓTESE DE PESQUISA

A subdivisão espacial de um problema de múltiplas classes desbalanceadas pode permitir a melhoria de desempenho dos algoritmos de classificação supervisionada. Esta subdivisão, com ajuste de representatividade ou não, pode descrever as regiões das classes, independentemente da quantidade de instâncias, de forma que até mesmo classificadores simples possam incrementar seu poder de generalização.

1.2 OBJETIVOS

Esta tese aborda problemas de classificação com múltiplas classes desbalanceadas. O principal objetivo é investigar como a disposição espacial dos dados pode influenciar no desempenho do modelo de classificação supervisionada utilizado.

Como forma de mitigação do problema avaliamos o uso de protótipos para a geração de dados das classes minoritárias durante o treinamento do modelo de classificação utilizado.

Além disso, como objetivo secundário, consideramos o impacto do desbalanceamento nos dados sobre as métricas de avaliação de algoritmos de classificação normalmente usadas na literatura.

1.3 CONTRIBUIÇÕES DA PESQUISA

A seguir são listadas algumas das contribuições do trabalho desenvolvido:

- Análises empíricas sugerindo que em uma base binária, um classificador sensível ao desbalanceamento está sujeito aos efeitos negativos do desbalanceamento quando a menor classe corresponder, em tamanho, a 25% ou menos da maior classe (SILVA; ZANCHETTIN, 2015);
- Análise de que a sobreposição de bordas é um dos principais causadores do efeito negativo do desbalanceamento (SILVA; ZANCHETTIN, 2015);
- Análise corroborando que as métricas *G-mean*, AUC e Acurácia Balanceada são apropriadas para o caso binário. Entretanto, a Acurácia Balanceada não se mostra apropriada quando estendida para múltiplas classes (SILVA; ZANCHETTIN, 2015; SILVA; ZANCHETTIN, 2016b);
- Análise de que classificadores podem se comportar de forma diferente em bases binárias e em bases com múltiplas classes. Esta alteração no comportamento está associada à disposição/distribuição das classes no espaço amostral (SILVA; ZANCHETTIN, 2016b);
- Estudo sobre a razão MIR (*Multiclass Imbalanced Ratio*) que demonstra o grau de heterogeneidade entre os tamanhos das classes, e pode ser usada como uma taxa de desbalanceamento para bases com múltiplas classes (SILVA; ZANCHETTIN, 2016b);
- Análises empíricas demostrando que algoritmos de geração de protótipos, apesar de simples, podem ser bastante eficientes na classificação de bases com múltiplas classes desbalanceadas (SILVA; ZANCHETTIN, 2016b). Com o ajuste da disposição espacial dos dados de treinamento, um classificador simples como o 1NN[2] pode alcançar desempenhos tão bons ou até superiores ao de comitês de classificadores;
- Análises sugerindo que a validação de experimentos se mostra mais adequada com o uso do método *10-fold* estratificado. Contudo, se há alguma classe extremamente rara na base de dados, o método *hold-out* 33% repetido se mostra o mais adequado (SILVA; ZANCHETTIN, 2016a);

[2] Vizinho mais próximo.

- Estudos que sugerem que uma classe pode ser considerada extremamente rara quando possuir menos de cinco instâncias (SILVA; ZANCHETTIN, 2016a). Desta forma é possível diferenciar classes raras de classes extremamente raras;

- Simplificação do *Gravity Search Algorithm* (GSA) (RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009), resultando no *Simple Gravity Search Algorithm* (SGSA) (SILVA; ZANCHETTIN, 2019). O conceito geral do algoritmo de busca é mantido, porém seu funcionamento é simplificado, resultando em desempenho similar, porém com a utilização de menos cálculos e parâmetros;

- Análises sugerindo que a normalização dos dados e seleção de atributos possuem influência positiva no desempenho em bases com múltiplas classes desbalanceadas (SILVA; ZANCHETTIN, 2019);

- Análises sugerindo que a geração dinâmica de protótipos é mais eficiente que a diminuição gradativa do conjunto de treinamento. Com o primeiro é possível a obtenção de desempenhos similares e melhores com uma quantidade menor de instâncias de treinamento (SILVA; ZANCHETTIN, 2019);

- Análises sugerindo que a utilização de mais de uma métrica de avaliação durante a geração de protótipos produz melhores desempenhos (SILVA; ZANCHETTIN, 2019).

Outras contribuições, podem ser consultadas nas seções 3.2.1, 3.4, e ao fim das seções 4.1, 4.2 e 4.3.

A pesquisa realizada também resultou nos seguintes artigos:

SILVA, E. J. R.; ZANCHETTIN, C. On the existence of a threshold in class imbalance problems. *IEEE International Conference on Systems, Man, and Cybernetics, 2015, Hong Kong*, p. 2714–2719, 2015. DOI:10.1109/SMC.2015.474.

SILVA, E. J. R.; ZANCHETTIN, C. A voronoi diagram based classifier for multiclass imbalanced data sets. *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, p. 109 – 114, 2016. Disponível em: <http://doi.org/10.1109/BRACIS.2016.030>.

SILVA, E. J. R.; ZANCHETTIN, C. On validation setup for multiclass imbalanced data sets. *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, p. 468–473, 2016. Disponível em: <https://doi.org/10.1109/BRACIS.2016.090>.

SILVA, E. J. R.; ZANCHETTIN, C. Dynamic centroid insertion and adjustment for data sets with multiple imbalanced classes. In: TETKO, I. V.; KŮRKOVÁ, V.; KARPOV, P.; THEIS, F. (Ed.). *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*. Cham: Springer International Publishing, 2019. p. 766–778. ISBN 978-3-030-30484-3. Disponível em: <https://doi.org/10.1007/978-3-030-30484-3_60>.

1.4 ORGANIZAÇÃO DO DOCUMENTO

A proposta de tese está organizada como segue.

O Capítulo 2 reúne a pesquisa bibliográfica. Nele estão os fundamentos e definições sobre o problema de classes desbalanceadas. O capítulo é dividido em quatro seções. Cada seção aborda um dos três principais tópicos de pesquisa. A última seção aborda o caso de múltiplas classes desbalanceadas.

O Capítulo 3 contém uma descrição detalhada do desenvolvimento e evoluções dos algoritmos propostos. O capítulo é dividido em quatro seções. A primeira apresenta a hipótese que deu origem ao estudo. A segunda seção apresenta detalhadamente a criação e evolução do algoritmo VDBC. A terceira seção apresenta a criação e evolução do algoritmo DCIA. Por fim, o capítulo é finalizado com um apanhado geral sobre o estudo conduzido.

O Capítulo 4 consiste nas comparações entre os algoritmos propostos e algoritmos da literatura que também abordam o problema de múltiplas classes desbalanceadas. O capítulo é dividido em três abordagens, nas quais o algoritmo proposto é comparado a algoritmos diferentes.

O Capítulo 5 contém as considerações finais e sugestões para trabalhos futuros.

O documento é finalizado com as seções de referências bibliográficas e apêndices com detalhes dos modelos propostos.

2 BASES DE DADOS COM CLASSES DESBALANCEADAS

O estudo em bases de dados com classes desbalanceadas pode ser visto como um tópico especial na área de Aprendizado de Máquinas e Mineração de Dados (PRATI; BATISTA; MONARD, 2004; LING; SHENG, 2008; NGUYEN; BOUZERDOUM; PHUNG, 2009; FARQUAD; BOSE, 2012; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). As classes de uma base de dados são consideradas desbalanceadas quando há uma diferença significativa entre as distribuições de suas amostras (BATISTA, 2003; CHAWLA; JAPKOWICZ; KOTCZ, 2004; NGUYEN; BOUZERDOUM; PHUNG, 2009; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LEE; LEE, 2012; LÓPEZ et al., 2013; MALDONADO; MONTECINOS, 2014; WANG, 2014). Em outras palavras, uma ou mais classes possuem uma quantidade bem menor de instâncias, em relação às demais classes. A detecção de fraudes é um exemplo de problema no qual o tamanho das classes difere significativamente. Outro exemplo é o diagnóstico médico, no qual o número de pacientes saudáveis é normalmente maior que o número de pacientes doentes.

Na literatura foi observada que a ocorrência de desbalanceamento entre as classes é comumente acompanhada de uma deterioração no desempenho dos classificadores. A queda do desempenho é associada ao enviesamento dos classificadores pela influência das maiores classes. Isto acontece devido aos cientistas normalmente assumirem que as classes estão balanceadas quando constroem os classificadores (BATISTA, 2003; NGUYEN; BOUZERDOUM; PHUNG, 2009). Desta forma, é possível que instâncias das classes menores sejam assumidas como ruído e, portanto, ignoradas. O enviesamento é também relacionado com a sobreposição de bordas entre as classes e a presença de classes raras (WEISS; PROVOST, 2001; JAPKOWICZ; STEPHEN, 2002; CHAWLA; JAPKOWICZ; KOTCZ, 2004; PRATI; BATISTA; MONARD, 2004; SILVA; ZANCHETTIN, 2015). Ambos os casos potencializam o problema.

Foi observado também que o desbalanceamento pode influenciar negativamente nos resultados obtidos com determinadas métricas. Essa influência é responsável por fazer com que o desempenho auferido seja melhor que o desempenho real do classificador (PRATI; BATISTA; MONARD, 2004; BRODERSEN et al., 2010; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LÓPEZ et al., 2013; HU; DONG, 2014; WANG, 2014).

Os estudos sobre o comportamento de classificadores em ambientes com dados desbalanceados podem ser divididos em três grandes linhas de pesquisa (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012):

1. Implementação de soluções para lidar com as bases desbalanceadas;
2. Métricas, i.e., estratégias para medir o desempenho dos algoritmos;
3. Análise de características que podem agravar o problema do desbalanceamento. Nesta linha são estudados também quais outros fatores podem influenciar no desempenho dos classificadores.

A continuação deste capítulo é organizada de acordo com essas linhas de pesquisa. As Seções 2.1, 2.2 e 2.3 se aprofundam sobre cada uma delas. A terceira linha é a primeira apresentada, de forma que o problema pode ser melhor entendido e vários termos técnicos necessários para as demais linhas são apresentados e explicados. As métricas e soluções implementadas para lidar com o desbalanceamento são apresentadas em seguida. Uma última seção (2.4) foi adicionada, para tratar do caso específico de bases de dados com múltiplas classes desbalanceadas.

2.1 INFLUÊNCIAS DO DESBALANCEAMENTO

Parte da pesquisa em bases de dados com classes desbalanceadas se dedica a entender a natureza do problema. Uma dúvida, por exemplo, é se classificadores são sempre influenciados pela diferença nas distribuições das classes. Independentemente da resposta a essa primeira dúvida, os pesquisadores procuram também investigar se existem outros fatores que possam influenciar negativamente no desempenho dos classificadores.

Os autores Japkowicz e Stephen (2002) são responsáveis por um dos trabalhos mais conhecidos nesta linha de pesquisa. Nele, eles procuram responder a três perguntas:

- Qual é a natureza do problema de classificação em problemas desbalanceados?
- Qual a diferença entre as abordagens para lidar com o problema?
- O problema das classes desbalanceadas prejudica o desempenho de outros classificadores, além do C5.0[1] investigado no estudo?

A primeira e a terceira perguntas se encaixam neste tópico. A segunda pergunta, entretanto, melhor se enquadra na Seção 2.3.

Japkowicz e Stephen (2002) sugerem que o desbalanceamento não possui influência quando as classes são linearmente separáveis (Figura 1). O problema de classes desbalanceadas apresenta-se, contudo, quando há certo nível de complexidade nos dados e é dependente, também, do tamanho do conjunto de treinamento.

A complexidade dos dados está associada a pequenos disjuntos (JAPKOWICZ; STEPHEN, 2002; NGUYEN; BOUZERDOUM; PHUNG, 2009; LÓPEZ et al., 2013) e pela sobreposição de bordas (NAPIERALA; STEFANOWSKI, 2012; PRATI; BATISTA; MONARD, 2004; MALDONADO; MONTECINOS, 2014). Os pequenos disjuntos são grupos de poucas instâncias que se encontram em alguma região cuja predominância é de outra classe (Figura 2). A sobreposição de bordas acontece em regiões fronteiriças entre as classes, na qual as instâncias estão dispostas ou misturadas de uma forma que se torna bastante difícil definir alguma fronteira entre os grupos (Figura 3).

[1] Este classificador é uma alteração feita sobre o C4.5 (QUINLAN, 1993).

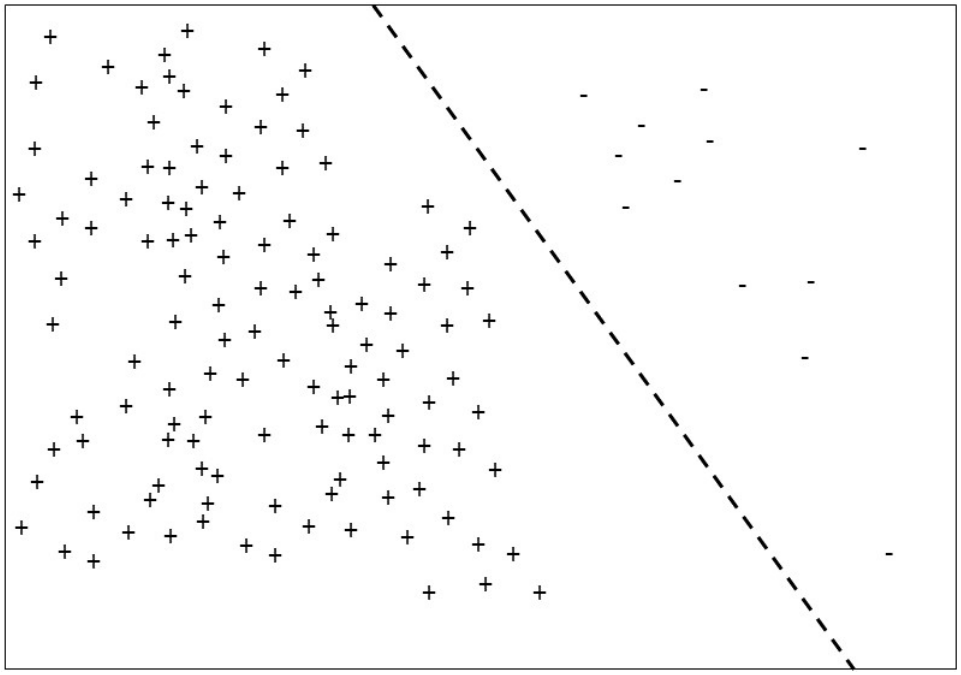

Figura 1 – Exemplo de uma base desbalanceada com classes linearmente separáveis.

A complexidade dos dados aumenta nesse caso porque se torna difícil distinguir os subgrupos de instâncias que podem ser consideradas ruidosas. A questão fica mais difícil quando os subgrupos estão nas bordas das classes, muitas vezes misturadas. Um algoritmo normalmente vai tratar os subgrupos da classe minoritária como ruído, uma vez que numa determinada região do espaço amostral, pode haver um claro domínio de uma classe sobre a outra (Figura 2). Portanto, havendo duas ou mais classes cujas bordas sejam complexas e sobrepostas (logo, menos discriminatórias), o desbalanceamento faz com que a classificação se torne uma tarefa mais desafiadora.

Aliado à complexidade dos dados está o tamanho do conjunto de treinamento. Havendo instâncias suficientes para o aprendizado dos conceitos das classes, o aprendizado não é tão desafiador, ainda que haja desbalanceamento (JAPKOWICZ; STEPHEN, 2002). Mesmo que as bordas das classes sejam complexas, o desempenho do classificador não é muito afetado. Entretanto, quanto menor a quantidade de instâncias de uma classe, mais difícil será o aprendizado de seu conceito e mais suscetível esta estará para os efeitos do desbalanceamento (Figura 4).

Quanto ao tamanho reduzido da quantidade de instâncias das classes, existem ainda dois outros fatores distintos que merecem atenção. O primeiro é o caso de classes extremamente raras, e o segundo é conhecido como *data set shift*.

Uma classe pode ser considerada extremamente rara quando possuir menos de cinco instâncias (SILVA; ZANCHETTIN, 2016a). Neste caso, tanto o conjunto de treinamento quanto o de teste podem ter instâncias insuficientes para que se garanta uma boa generalização do conceito da classe, ou até mesmo a avaliação desta generalização. Mesmo téc-

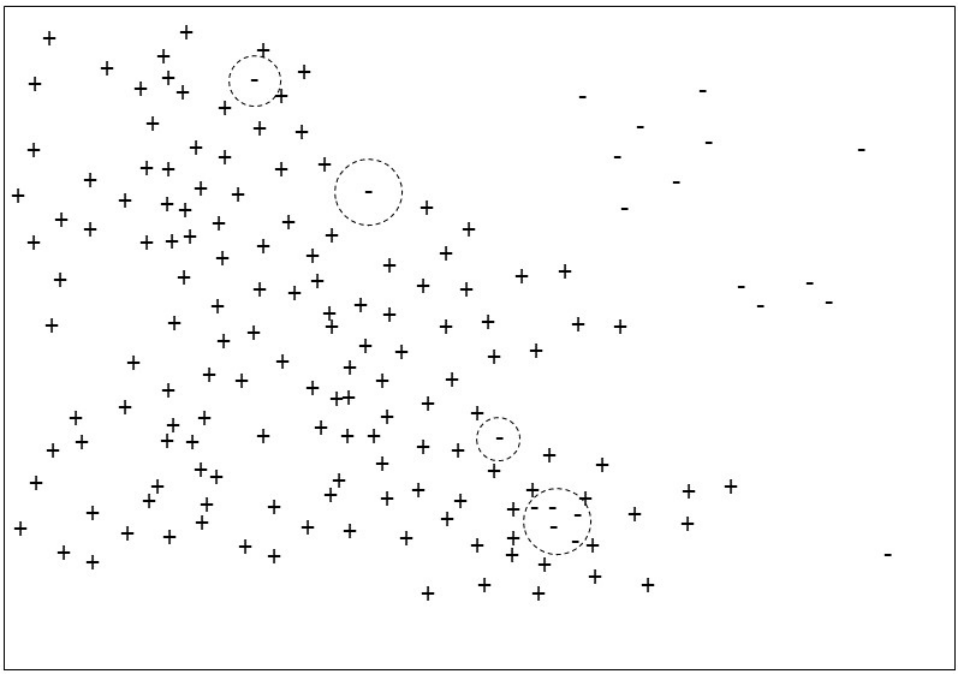

Figura 2 – Exemplo de pequenos disjuntos e instâncias de uma classe minoritária que seriam consideradas como ruído da classe majoritária.

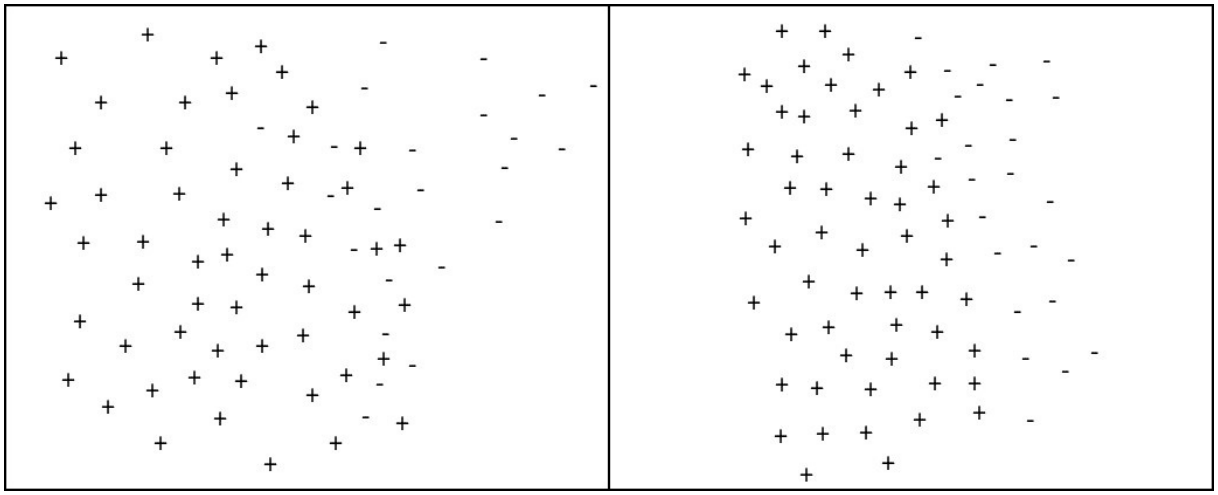

Figura 3 – Exemplo de uma base desbalanceada com bordas sobrepostas, à esquerda, e sem sobreposição de bordas, à direita.

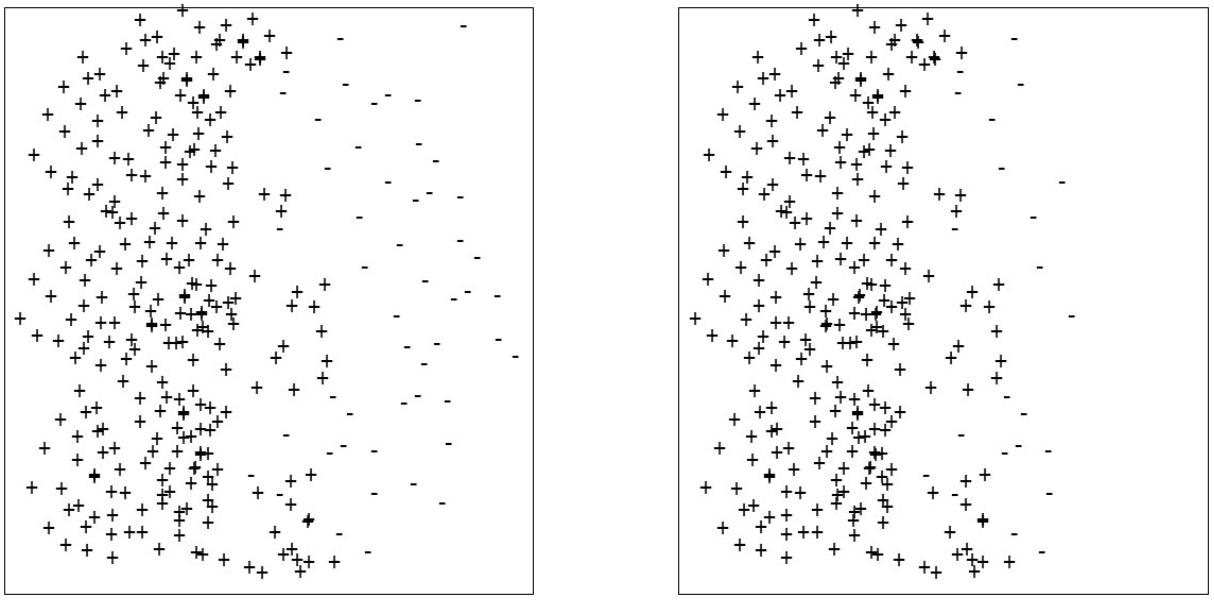

Figura 4 – Exemplo da influência do tamanho da classes no conjunto de treinamento. No exemplo à esquerda, mesmo havendo desbalanceamento, um classificador provavelmente conseguirá generalizar para a classe minoritária. O mesmo se torna mais difícil no exemplo à direita.

nicas de sobreamostragem podem ser insuficientes, dado o número reduzido de instâncias conhecidas, por exemplo, quando uma classe tiver apenas uma instância de treinamento e uma, ou nenhuma, instância de teste. Ainda, quanto menor a quantidade de instâncias de classe, mais significativo será cada erro. Por exemplo, caso um algoritmo classifique corretamente apenas uma de duas instâncias no conjunto de teste, sua acurácia para esta classe será de 50%.

É importante notar que existe uma diferença entre classe rara e classe extremamente rara. No primeiro caso, pode acontecer da classe minoritária ter uma grande quantidade de instâncias, entretanto, a classe majoritária ainda tem uma quantidade consideravelmente maior. Ou seja, uma classe pode ser considerada rara devido à quantidade de instâncias da(s) outra(s) classe(s). No segundo caso, a quantidade de instâncias é tão pequena que afeta os experimentos. Por exemplo, na divisão dos dados em conjunto de treinamento e teste, a classe minoritária pode ficar ausente de algum conjunto. Ainda, como referido anteriormente, a análise do desempenho do classificador, levando-se em conta tais classes pode ser dificultada ou prejudicada. Ou seja, classes extremamente raras são aquelas que, de fato, possuem poucas instâncias.

O *data set shift* consiste em quando as instâncias de treinamento e as de teste refletem distribuições diferentes (Figura 5) (LÓPEZ et al., 2013). Neste caso, mesmo que a classe minoritária não seja rara, ao ser dividida em dois conjuntos, pode ser considerada como duas classes diferentes por um determinado algoritmo. O problema pode piorar quando uma base é desbalanceada (LÓPEZ; FERNÁNDEZ; HERRERA, 2014).

Os três tipos mais conhecidos de *data set shift* são (MORENO-TORRES et al., 2012;

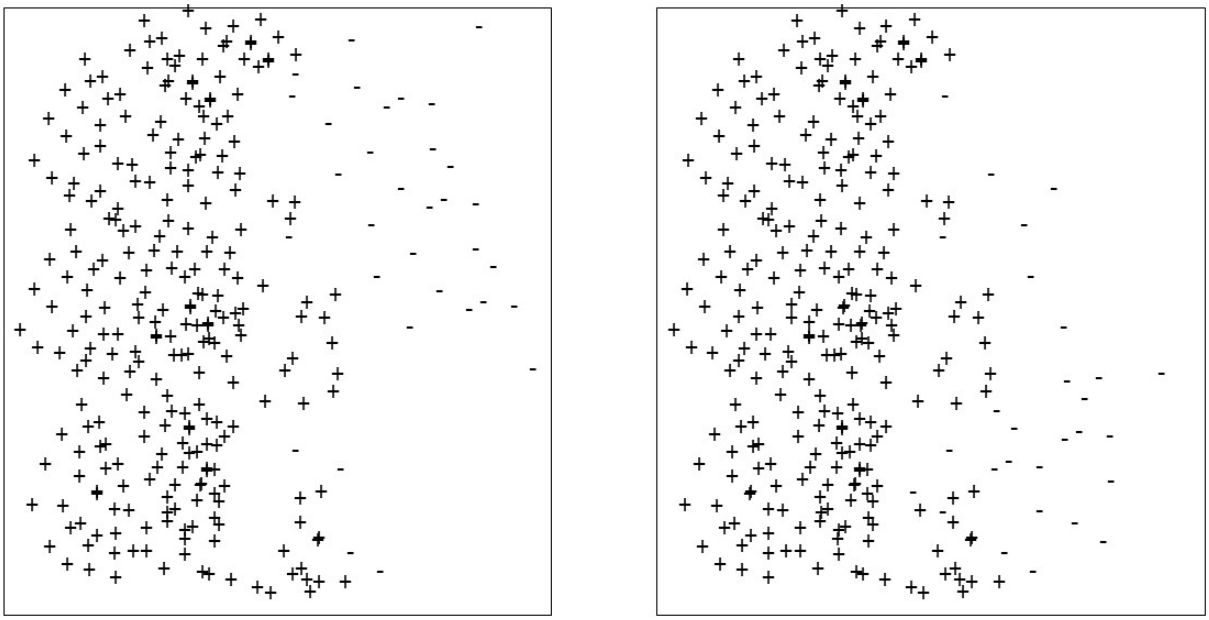

Figura 5 – Exemplo de *data set shift* nas instâncias da classe minoritária (a classe majoritária não foi separada).

LÓPEZ; FERNÁNDEZ; HERRERA, 2014):

- *Prior Probability Shift*, ou *mudança de probabilidade* a priori. Este é o caso no qual as probabilidades das classes são modificadas. Por exemplo, se as probabilidades das classes forem 70% e 30%, respectivamente, porém no conjunto de treinamento e teste essas probabilidades estão diferentes. Uma maneira de se evitar este problema é dividir os dados de forma estratificada;

- *Covariate Shift*, ou *mudança de covariável*. Neste caso, o conceito de uma classe está presente em partes nos conjuntos de treinamento e teste. Desta forma, um classificador pode aprender que determinada região do espaço amostral não seja de uma classe, por falta de exemplos. Durante o teste, as instâncias dessa região serão erroneamente classificadas;

- *Concept Shift*, ou *mudança de conceito*. É o caso mais desafiador, pois o conceito de uma classe é completamente diferente entre os conjuntos de treinamento e teste.

Por fim, quanto aos classificadores, alguns podem ser considerados como *naturalmente* sensíveis ao desbalanceamento (JAPKOWICZ; STEPHEN, 2002; SILVA; ZANCHETTIN, 2015). Classificadores como Árvores de Decisão e *Multilayer Perceptron* (MLP) (HAYKIN, 1999) perdem em desempenho quando o tamanho da classe minoritária corresponde a 25%, ou menos, do tamanho da classe majoritária, mesmo que não haja sobreposição de bordas (SILVA; ZANCHETTIN, 2015). Por outro lado, classificadores como *Support Vector Machine* (SVM) demonstram ser praticamente insensíveis ao desbalanceamento, sofrendo pouca

ou nenhuma influência negativa em bases binárias (JAPKOWICZ; STEPHEN, 2002; SILVA; ZANCHETTIN, 2015).

2.2 MÉTRICAS DE AVALIAÇÃO DE DESEMPENHO

A avaliação do desempenho de algum algoritmo é realizada com a utilização de uma função de análise ou métrica de avaliação. Através dos valores obtidos por meio de uma métrica de avaliação, é possível compreender a eficiência de um algoritmo em uma determinada tarefa. Com esses valores é possível também comparar algoritmos diferentes, e então determinar se o desempenho de algum é superior ao do outro. Da mesma forma, é possível observar a evolução das técnicas ao longo do tempo, à medida em que as métricas atestam sua melhora ou piora no desempenho. Ao mesmo tempo, uma métrica indevida pode levar a análises incorretas sobre o desempenho de um algoritmo. Portanto, a escolha de uma métrica apropriada é importante no processo de avaliação de algoritmos.

Em Aprendizado de Máquina, duas das métricas mais utilizadas são a *acurácia* e a *taxa de erro*. A acurácia é calculada ao se dividir o número de classificações corretas pelo número total de instâncias. Desta forma, a acurácia pode também ser referida como *taxa de acerto* [*geral*]. Os valores possíveis para a acurácia variam entre 0 e 1, ou entre 0% e 100%, de forma que 0 significa que o algoritmo errou a classificação de todas as instâncias. O valor 1 significa que o algoritmo acertou a classificação de todas as instâncias. A *taxa de erro* é calculada como: $1 - acurácia$.

Entretanto, ao tratar de bases de dados com classes desbalanceadas, a acurácia — e por consequência a taxa de erro — podem se tornar uma métrica enganosa (PRATI; BATISTA; MONARD, 2004; BRODERSEN et al., 2010; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LÓPEZ et al., 2013; HU; DONG, 2014; WANG, 2014). Por exemplo, suponha uma base de dados binária altamente desbalanceada, na qual uma das classes possui 990 instâncias, enquanto a outra possui apenas 10 instâncias. Se um algoritmo classificar qualquer instância recebida como pertencente à primeira classe, o mesmo conseguiria, neste caso, uma acurácia de 0,99 ou 99% de acerto. Isto faz com que se tenha uma falsa impressão de que o algoritmo foi eficiente na inferência dos dados como um todo. Logo, percebe-se que este valor, apesar de ter sido calculado de forma correta, não expressa corretamente o fato de que uma das classes foi erroneamente classificada por completo.

Portanto, uma vez que a acurácia é facilmente influenciada pelas classes maiores de uma base de dados, é necessária a utilização de outra métrica que lide de forma adequada com classes desbalanceadas. Uma métrica apropriada, neste caso, deve ser capaz de lidar de forma proporcional com as classes (SILVA; ZANCHETTIN, 2015).

As métricas normalmente são calculadas a partir das informações extraídas de uma Matriz de Confusão (Tabela 1). Em uma base de dados binária e desbalanceada, a maior classe é referida como *majoritária*. Ao mesmo tempo, a menor classe é referida como *minoritária*. Na matriz de confusão, a classe minoritária é referida, na literatura, como a

classe positiva, enquanto a classe majoritária é referida como a classe negativa (NAPIERALA; STEFANOWSKI, 2012). A razão disso é que normalmente a classe minoritária é a de maior importância no contexto do problema tratado. Uma vez que a base é binária, a classificação pode seguir o estilo *booleano*. Portanto as instâncias são classificadas como pertencente ou não à classe de interesse, que, neste caso, é a minoritária.

Tabela 1 – Matriz de Confusão.

	Predição Positiva	**Predição Negativa**
Classe Positiva	Verdadeiro Positivo (VP)	Falso Negativo (FN)
Classe Negativa	Falso Positivo (FP)	Verdadeiro Negativo (VN)

As métricas *clássicas* que são extraídas diretamente da matriz de confusão são: Acurácia (*Accuracy*), Sensibilidade (*Sensitivity* também conhecida como *Recall*), Especificidade (*Specificity*) e Precisão (*Precision*). A partir dessas é possível derivar as métricas Média Geométrica (*G-mean*), *F-measure*, F_1, Acurácia Balanceada (*Balanced Accuracy*) (BRODERSEN et al., 2010), *Optimized Precision* (OP) (RANAWANA; PALADE, 2006) e *[Generalized] Index of Balanced Accuracy* (IBA) (GARCÍA; MOLLINEDA; SANCHEZ, 2010).

$$\text{Acurácia} = \frac{VP + VN}{VP + FP + FN + VN} \tag{2.1}$$

$$\text{Sensibilidade} = \text{Recall} = \frac{VP}{VP + FN} \tag{2.2}$$

$$\text{Especificidade} = \frac{VN}{FP + VN} \tag{2.3}$$

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2.4}$$

A Sensibilidade (2.2), que além de *Recall* também é conhecida como Taxa de Verdeiro Positivo (TVP), é a porcentagem de instâncias positivas que são corretamente classificadas. Pode ser referida como a acurácia da classe positiva. De forma similar, a Especificidade (2.3) — ou Taxa de Verdadeiro Negativo (TVN) — é a porcentagem de instâncias negativas que foram corretamente classificadas. Pode também ser referida como acurácia da classe negativa.

A métrica Precisão (2.4) é focada na *pureza* da classificação positiva (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012), e expressa o desempenho do classificador em relação a apenas uma classe. Esta métrica é a porcentagem de acerto de instâncias positivas sobre todas as instâncias que foram classificadas como positivas.

$$\textit{G-mean} = \sqrt{\textit{Sensibilidade} \times \textit{Especificidade}} \tag{2.5}$$

$$F\text{-}measure = \frac{(1+\beta^2) \times (Precisão \times Sensibilidade)}{\beta^2 \times Precisão + Sensibilidade} \tag{2.6}$$

$$F_1 = \frac{2 \times (Precisão \times Sensibilidade)}{Precisão + Sensibilidade} \tag{2.7}$$

A média geométrica (KUBAT; MATWIN, 1997), ou *G-mean* (2.5) é calculada a partir do produto das acurácias de cada classe, e subsequentemente o valor é elevado ao inverso da quantidade de classes. Esta métrica indica o balanceamento entre os desempenhos obtidos sobre as diferentes classes. Portanto, um baixo desempenho de classificação sobre qualquer classe será refletido sobre esta métrica.

A métrica *F-measure* (2.6) é a média harmônica ponderada entre Precisão e Sensibilidade. O parâmetro β controla a influência da Precisão e da Sensibilidade separadamente (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). Quando $\beta = 0$, a *F-measure* se torna equivalente à Precisão. Por outro lado, *F-measure* se aproxima da Sensibilidade quando $\beta \to \infty$. Na literatura, entretanto, é comum $\beta = 1$. Desta forma *F-measure* se transforma na média harmônica entre Precisão e Sensibilidade, passando a ser referida como F_1 (2.7).

$$\text{Acurácia Balanceada} = \frac{Sensibilidade + Especificidade}{2} \tag{2.8}$$

A Acurácia Balanceada (2.8) é literalmente a média aritmética entre as taxas de acerto de ambas as classes. Entretanto, caso o pesquisador deseje atribuir um peso diferente às taxas de acerto, basta utilizar a equação (2.9). De fato, a equação (2.8) equivale à equação (2.9) com $\alpha = 0,5$.

$$\text{Acurácia Balanceada} = \alpha \times Sensibilidade + (1-\alpha) \times Especificidade \tag{2.9}$$

$$OP = \text{Acurácia} - \frac{|Especificidade - Sensibilidade|}{Especificidade + Sensibilidade} \tag{2.10}$$

A métrica OP (2.10) surgiu como uma tentativa de remediar uma deficiência presente em métricas como *G-mean* (2.5). No caso desta última métrica, não é possível distinguir a contribuição que cada classe teve para o desempenho geral do modelo, ou ainda qual das classes é a majoritária (ou minoritária). Em outras palavras, é possível obter o mesmo desempenho com combinações diferentes entre Sensibilidade e Especificidade. A obtenção de um alto valor para OP requer um alto valor de acurácia e, ao mesmo tempo, classes pouco desbalanceadas. Entretanto é possível que o valor da acurácia seja alto enquanto as classes são bastante desbalanceadas. Este fator pode influenciar fortemente no valor final da métrica OP (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). Ainda, caso o interesse na classe minoritária seja o bastante para destacá-la, basta ao pesquisador seguir o *padrão informal* sugerido na literatura, na qual a classe minoritária é normalmente associada à classe positiva, e utilizar a métrica Acurácia Balanceada em conjunto com as métricas

Sensibilidade e Especificidade (SILVA; ZANCHETTIN, 2015), ou a Precisão (LÓPEZ et al., 2013).

$$IBA_{\alpha}(\mathcal{M}) = (1 + \alpha \cdot Dom) \cdot \mathcal{M} \tag{2.11}$$

Na equação (2.11), $\mathcal{M}$ é uma métrica. Dom é a dominância, definida como $Dom = Sensibilidade - Especificidade$, no intervalo $[-1, +1]$. A dominância é ponderada por $\alpha \geq 0$, para reduzir sua influência sobre a métrica $\mathcal{M}$. A função IBA quantifica um equilíbrio entre uma medida geral de desempenho (obtida através de $\mathcal{M}$) e um índice do quão balanceada estão as acurácias das classes (obtido através de $(1 + \alpha \cdot Dom)$), pois a tendência é que quanto mais desbalanceadas são as classes, mais discrepantes serão os valores de Sensibilidade e Especificidade. O objetivo é favorecer moderadamente os algoritmos de classificação com melhor desempenho na classe minoritária, sem subestimar a relevância da classe majoritária (LÓPEZ et al., 2013). Em García, Sánchez e Mollineda (2012), os autores definem $\alpha = 0,01$ e $\mathcal{M} = Gmean^2$.

Outras duas métricas que recebem bastante atenção na literatura são *Receiver Operating Characteristics* (ROC) e *Area Under ROC Curve* (AUC) (LING; HUANG; ZHANG, 2003). A curva ROC consiste em visualizar a taxa de verdadeiros positivos como uma função da taxa de falsos positivos sobre todos os valores possíveis para o classificador (WANG, 2014). Um exemplo de uma curva ROC em um plano cartesiano bidimensional é apresentado na Figura 6. No eixo das ordenadas é mapeada a taxa de verdadeiros positivos. No eixo das abscissas é mapeada a taxa de falsos positivos. O ponto (0, 0) equivale a um algoritmo que classifique todas as instâncias como negativas. O ponto (1, 1) equivale a um algoritmo que classifique todas as instâncias como positivas. O ponto (0, 1) equivale a um algoritmo que classifique corretamente todas as instâncias.

A análise e a comparação entre vários algoritmos utilizando da curva ROC pode ser facilitada pela métrica AUC (LING; HUANG; ZHANG, 2003; WANG, 2014), que é o cálculo da área sob a curva ROC. Quanto maior o valor do AUC, melhor será o desempenho de um classificador, uma vez que a curva se aproxima do ponto (0, 1). Em uma base binária, entretanto, o valor do AUC será idêntico ao valor da Acurácia Balanceada (GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; HU; DONG, 2014; SILVA; ZANCHETTIN, 2015), como na equação (2.9), com $\alpha = 0,5$.

Uma maneira de se calcular o AUC é apresentada na equação (2.12). TVP se refere à Taxa de Verdadeiro Positivo, enquanto TFP se refere à Taxa de Falso Positivo, ou seja, $1 - Especificidade$.

$$AUC = (1 + TVP - TFP)/2 \tag{2.12}$$

Muitos pesquisadores preferem utilizar AUC ou *G-mean*, o que faz com que ambas sejam facilmente encontradas na literatura. Entretanto, é bastante comum que cada trabalho utilize uma métrica diferente. Esta falta de consenso faz com que a escolha da

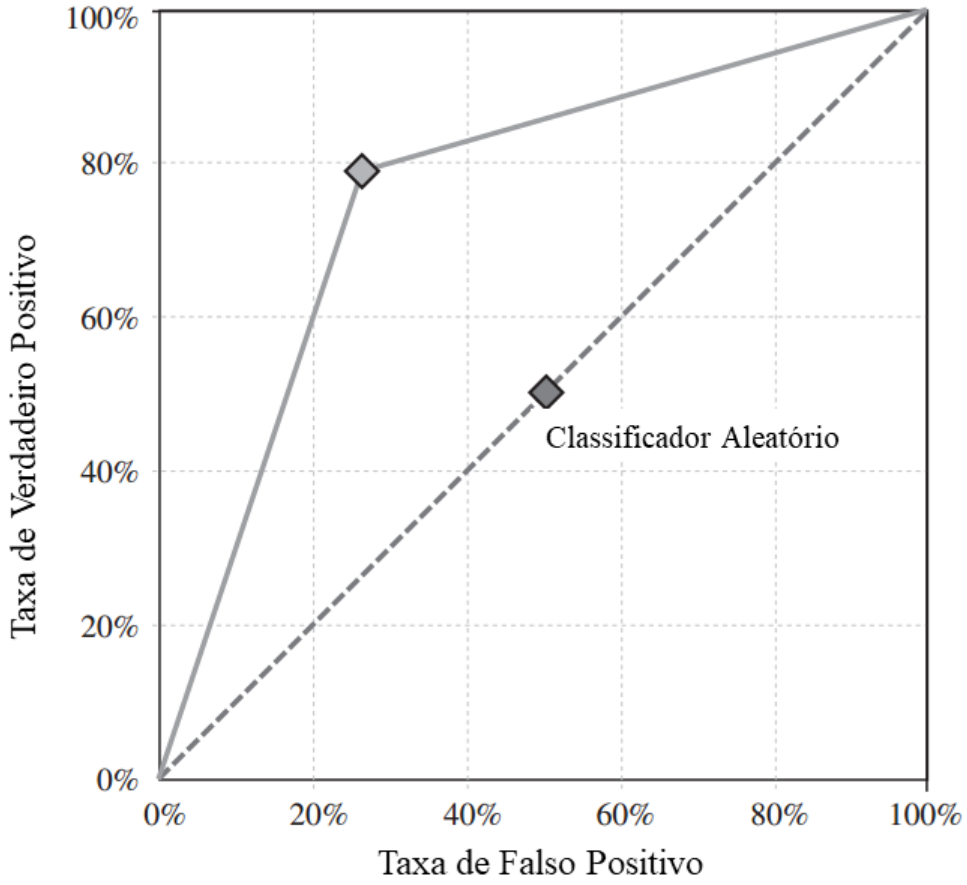

Figura 6 – Exemplo de curva ROC. A linha tracejada representa um classificador aleatório, enquanto que a linha sólida representa um classificador cujo desempenho é melhor que a classificação aleatória. *True Positive Rate* é a taxa de verdadeiros positivos e *False Positive Rate* é a taxa de falsos positivos (fonte: (LÓPEZ et al., 2013)).

métrica ainda seja um problema ao lidar com bases de dados com classes desbalanceadas, especificamente na questão das comparações entre algoritmos.

Por fim, existe ainda outra métrica bastante utilizada na área, cujo foco está na base de dados em vez do desempenho de algum algoritmo. Esta métrica, conhecida como *Imbalance Ratio* (IR) ou Taxa de Desbalanceamento, calcula o quanto uma base binária é desbalanceada ao dividir o tamanho da maior classe pelo tamanho da menor classe.

2.3 ABORDAGENS PARA LIDAR COM O DESBALANCEAMENTO

As abordagens para bases de dados com classes desbalanceadas podem ser divididas em dois grandes grupos:

- **Nível de dados**: manipulação direta dos dados via reamostragem. Na maioria dos casos a manipulação é feita de duas formas, sobreamostragem ou subamostragem. Existem ainda os trabalhos que investigam a utilização de ambas as técnicas de

reamostragem. Uma terceira forma de manipulação dos dados é a utilização de técnicas de redução de instâncias;

- **Nível de algoritmo**: adaptação de classificadores e modelos de aprendizagem a fim de ajustá-los ao problema. Estes algoritmos podem ser referenciados como *ad hoc*, uma vez que são modificações ou criações de classificadores, ou modelos de classificação, feitas especialmente para lidar com o desbalanceamento.

Existem ainda outros tipos de abordagem, e.g., Aprendizado Sensível ao Custo, *One-Class Classifier* e Comitês de Classificadores. Alguns autores preferem identificá-los como variações da abordagem em nível de algoritmo (GARCÍA et al., 2009; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012). Outros autores, entretanto, preferem referenciá-los independentemente (NGUYEN; BOUZERDOUM; PHUNG, 2009; LÓPEZ et al., 2013). Neste trabalho, preferiu-se considerar esses demais tipos de abordagens como soluções em nível de algoritmo, uma vez que consistem em adaptações de classificadores.

2.3.1 Abordagem em nível de dados

O objetivo básico das técnicas de reamostragem (*resampling*) é balancear a quantidade de instâncias entre as classes. As duas formas mais comuns de se balancear as classes são sobreamostragem (*oversampling*) e subamostragem (*undersampling*). A primeira busca criar novas instâncias para a classe minoritária, i.e., a classe que possui a menor quantidade de instâncias. Por outro lado, a subamostragem procura excluir instâncias da classe majoritária, ou seja, a classe com maior quantidade de instâncias.

2.3.1.1 Sobreamostragem

A sobreamostragem pode acontecer de duas formas: replicação das instâncias existentes ou criação de instâncias sintéticas (NGUYEN; BOUZERDOUM; PHUNG, 2009; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LÓPEZ et al., 2013; WANG, 2014; MALDONADO; MONTECINOS, 2014). A mais simples das técnicas de sobreamostragem é a Sobreamostragem Aleatória, ou *Random Oversampling* (ROS), na qual instâncias de uma classe são selecionadas aleatoriamente e replicadas até que se atinja um nível satisfatório de balanceamento entre as classes. A vantagem dessa técnica é o balanceamento entre as classes sem a adição de novas informações (gerar novas informações aleatórias pode ser prejudicial à distribuição dos dados). Entretanto, uma quantidade elevada de instâncias idênticas, além de aumentar o custo computacional, pode fazer com que classificadores passem a considerar apenas determinados pontos, e não a região em sua volta, como área da classe. Desta forma, pode haver perda de generalização, ou seja, *overfitting*.

Outra técnica de sobreamostragem bastante conhecida é o *Synthetic Minority Oversampling Technique* (SMOTE) (CHAWLA et al., 2002). Esta técnica permite a criação de

instâncias sintéticas ao interpolar as instâncias de uma mesma classe que sejam vizinhas no espaço amostral. Desta forma, é possível aumentar a quantidade de instâncias de uma classe pequena com a adição de informação, resolvendo — ao menos em parte — o problema de *overfitting*. Entretanto, a utilização do SMOTE pode resultar no aumento da sobreposição de bordas entre instâncias de diferentes classes (LÓPEZ et al., 2013).

A partir do algoritmo original, algumas modificações ao SMOTE foram sugeridas. Entre elas pode-se citar o SMOTEBoost, que aplica SMOTE a cada iteração do algoritmo (CHAWLA et al., 2003); *Borderline*-SMOTE, que aplica SMOTE somente às instâncias da classe minoritária que estejam na borda (HAN; WANG; MAO, 2005); MSMOTE, que ignora as instâncias da menor classe, as quais são consideradas como ruído (HU et al., 2009); e *Safe Level* SMOTE (BUNKHUMPORNPAT; SINAPIROMSARAN; LURSINSAP, 2009), que aplica SMOTE somente às instâncias consideradas seguras.

Outros algoritmos de sobreamostragem além do SMOTE são: *Adaptive Synthetic sampling approach* (ADASYN), que cria adaptativamente mais amostras sintéticas para instâncias difíceis de serem classificadas (HE et al., 2008); e RWO-Sampling, que cria instâncias sintéticos aplicando *random walk* (ZHANG; LI, 2014).

2.3.1.2 Subamostragem

A subamostragem acontece com a exclusão de instâncias do conjunto de treinamento (NGUYEN; BOUZERDOUM; PHUNG, 2009; GARCÍA; SÁNCHEZ; MOLLINEDA, 2012; LÓPEZ et al., 2013; MALDONADO; MONTECINOS, 2014; WANG, 2014). Esta forma de reamostragem possui a vantagem de diminuir o custo computacional, uma vez que os classificadores lidarão com um conjunto menor de dados. Entretanto a subamostragem possui o risco de excluir dados valiosos, ou seja, instâncias que sejam discriminatórias e, por isso, poderiam influenciar algum classificador a melhorar o seu desempenho.

A técnica mais simples de subamostragem é a Subamostragem Aleatória, ou *Random Undersampling* (RUS). Neste caso, instâncias de uma classe específica são selecionadas ao acaso e retiradas do conjunto de treinamento até que se atinja um nível desejado de balanceamento.

Outros algoritmos também se encaixam nesta técnica. Pode-se citar: *One-Sided Sampling* (OSS), que exclui as instâncias da maior classe que sejam redundantes ou próximas à borda (KUBAT; MATWIN, 1997); *Neigborhood Cleaning rule* (NCL), que retira instâncias da maior classe que sejam ruído ou que façam parte dos três vizinhos mais próximos de uma instância da menor classe que foi erroneamente classificada (LAURIKKALA, 2001); *[under]Sampling Based on Clustering* (SBC), que divide as instâncias em k *clusters*, determina a quantidade de instâncias da maior classe que podem ser selecionadas em cada *cluster* e separa aleatoriamente essa quantidade de cada *cluster* (YEN et al., 2006); *Granular SVMs-Repetitive Undersampling* (GSVM-RU), que forma o subconjunto de instâncias da maior classe ao encontrar vetores de suporte ao longo de várias iterações (TANG et al.,

2009); *Weighted Undersampling*, que para cada instância da menor classe calcula a distância ponderada das instâncias da maior classe, e seleciona uma quantidade de instâncias de acordo com uma taxa de balanceamento pré-definida (ANAND et al., 2010).

Diversos outros algoritmos de subamostragem podem ser encontrados na literatura. Os algoritmos citados são apenas exemplos, uma vez que não faz parte do escopo deste trabalho uma listagem exaustiva dos modelos existentes.

2.3.1.3 Combinações de reamostragem

Em algumas situações pode ser vantajosa a utilização de ambas as técnicas de reamostragem para o pré-processamento dos dados. Neste caso, tanto a classe minoritária quanto a majoritária sofrem alterações com o objetivo de melhorar o desempenho de algum classificador.

Por exemplo, Estabrooks, Jo e Japkowicz (2004) constroem um comitê de classificadores, no qual cada classificador é treinado em um tipo de reamostragem (tanto sobreamostragem, quanto subamostragem), e selecionam para cada instância de teste um dos classificadores.

Liu, An e Huang (2006) utilizam SMOTE na menor classe e, posteriormente, aplicam a técnica *bootstrap* para excluir instâncias da maior classe. Após o processo de reamostragem uma SVM é treinada.

O algoritmo *Selective Preprocessing of Imbalanced Data* (SPIDER) (STEFANOWSKI; WILK, 2008) implementa a reamostragem em duas fases. Na primeira fase as instâncias são rotuladas como *ruidosas* ou *seguras*. Na segunda fase ocorre a subamostragem da maior classe, excluindo as instâncias que sejam ruidosas e que estejam muito próximas de instâncias ruidosas da menor classe. Após esse processo acontece a sobreamostragem da menor classe.

Por fim, Napierala, Stefanowski e Wilk (2010) introduzem o algoritmo SPIDER2. Neste algoritmo a rotulação das instâncias é feita em um pré-processamento e as fases correspondem à subamostragem e sobreamostragem, respectivamente. Outra diferença, em relação ao SPIDER, é que a subamostragem ocorre não só ao excluir instâncias como também ao modificar a classe de algumas instâncias da maior classe, passando a considerá-los como exemplos da menor classe.

2.3.1.4 Redução de Instâncias

A redução de instâncias como uma abordagem em nível de dados pode ser considerada como uma terceira via, ainda pouco explorada. Sua particularidade reside no fato de que todo o conjunto de treinamento é transformado, resultando em um conjunto menor contendo apenas protótipos. Um protótipo é uma instância que representa um conjunto de exemplos de uma determinada região do espaço amostral. Normalmente, os exemplos representados são todos da mesma classe.

Os métodos, ou técnicas de redução de instância têm por objetivo encontrar o melhor conjunto reduzido de instâncias que represente o conjunto de treinamento original (LÓPEZ et al., 2014). Estes métodos podem ser divididos em Seleção de Protótipos (SP) e Geração de Protótipos (GP) (LÓPEZ et al., 2014). O primeiro procura selecionar um subconjunto do conjunto original de treinamento, enquanto o segundo constrói um novo conjunto de instâncias, o qual pode ser completamente diferente do original.

O *Fuzzy Rough Imbalanced Prototype Selection* (FRIPS) (VERBIEST et al., 2012) é um exemplo de algoritmo desta categoria. Primeiro, é calculado o nível de ruído para cada instância, cujos valores são tratados como limites. Para cada limite é possível selecionar um subconjunto de instâncias cujos níveis de ruído são menores que o limite em questão. Cada subconjunto de treinamento é classificado com 1NN, e o subconjunto associado ao melhor desempenho é selecionado para ser utilizado pelo 1NN no conjunto de testes.

Outro exemplo é o algoritmo *Adaptive Self-Generating Prototype* (ASGP) (OLIVEIRA et al., 2012), que consiste em uma adaptação do SGP2 (*Self-Generating Prototype*) (FAYED; HASHEM; ATIYA, 2007) para bases desbalanceadas. O SGP, cuja variante é o SGP2, é um algoritmo de GP que encontra a quantidade de protótipos e suas localizações a partir de um processamento sobre o conjunto de treinamento. Os autores do ASGP argumentam que, apesar de ser um algoritmo bastante eficiente, durante sua execução o SGP2 acaba considerando instâncias da classe minoritária como ruídos ou *outliers* que precisam ser removidos. Logo, os passos prejudiciais à classe minoritária foram remodelados para se tornarem adaptativos em relação à quantidade de instâncias de cada classe. Posteriormente, foi descoberto que o ASGP gera mais protótipos para a classe minoritária que o necessário. Para lidar com este novo problema, os autores sugeriram o algoritmo *Evolutionary Adaptive Self-Generating Prototype* (EASGP) como solução (OLIVEIRA et al., 2015).

Por fim, López et al. (2014) propõem o *Iterative Prototype Adjustment Based on Differential Evolution for Imbalanced Domains* (IPADE-ID), adaptação do IPADE (TRIGUERO; GARCÍA; HERRERA, 2010) para bases desbalanceadas. O algoritmo consiste em três fases: inicialização, bastante similar ao SGP; ajuste do posicionamento dos protótipos através de uma busca com a Evolução Diferencial (STORN; PRICE, 1997); e a utilização de SP para a inserção de novos protótipos.

2.3.2 Abordagem em nível de algoritmo

Os classificadores *ad hoc* — no contexto deste trabalho — são aqueles modificados ou implementados com o propósito de lidar com o desbalanceamento entre as classes de uma base de dados.

Em bases de dados com classes desbalanceadas alguns classificadores sofrem bastante influência das classes maiores (BATISTA, 2003; NGUYEN; BOUZERDOUM; PHUNG, 2009). Isto se deve ao fato de os classificadores serem treinados para melhorar sua acurácia geral.

Neste caso, quanto menor é uma classe, menor será sua influência no desempenho geral. Uma solução para esses casos é modificar os classificadores para que os mesmos possam lidar com o desbalanceamento.

Por outro lado, existem classificadores que são pouco afetados pelo desbalanceamento, como é o caso das SVMs (JAPKOWICZ; STEPHEN, 2002; SILVA; ZANCHETTIN, 2015). Logo, algumas de suas características podem ser melhoradas para um aumento de desempenho, ou replicadas para que outros classificadores também atinjam melhores desempenhos.

Como exemplos de classificadores *ad hoc*, pode-se citar: *Hellinger Distance Decision Tree* (HDDT) (CIESLAK; CHAWLA, 2008), que utiliza a Distância Hellinger, insensível ao desbalanceamento, como critério de divisão de uma árvore de decisão; uma nova função de erro para o algoritmo *Backpropagation*, de forma a intensificar a atualização dos pesos para a classe minoritária enquanto a atualização de pesos para a classe majoritária é atenuada (OH, 2011); e *Weighted ELM* (WELM) (ZONG; HUANG; CHEN, 2013) que propõe dar maior peso à classe minoritária e, desta forma, modifica a borda de classificação produzida pelo *Extreme Learning Machine* (ELM).

Apesar destes classificadores modificados, ou melhorados, normalmente não utilizarem pré-processamento nos dados, é possível que técnicas de reamostragem também sejam usadas nesse contexto, como parte de um processo maior. Exemplos dessa combinação são o SMOTEBoost (CHAWLA et al., 2003) e a proposta de Nguyen, Bouzerdoum e Phung (2009). Quanto ao SMOTEBoost, é interessante ressaltar que este algoritmo também faz parte dos comitês de classificadores. Outro exemplo de comitês de classificadores utilizado em bases desbalanceadas pode ser encontrado em (MALDONADO; MONTECINOS, 2014). Neste trabalho os autores verificam a utilização de comitês de SVMs e *Support Vector Data Description* (SVDD), i.e., um classificador de uma classe (*one-class classifier*), em bases extremamente desbalanceadas, com sobreposição de bordas e ruído.

Por fim, merece destaque também a utilização de Redes Neurais Profundas (RNP), ou *Deep Neural Networks* (DNN), no contexto de bases desbalanceadas. Até então este campo tem sido pouco explorado, e em um *survey* recente foram destacados somente quinze trabalhos relevantes (JOHNSON; KHOSHGOFTAAR, 2019). A maior parte destes trabalhos possuem um foco claro em adaptar *Convolutional Neural Networks* (CNN) para o processamento de imagens. Isto pode ser notado na página 48 do artigo (JOHNSON; KHOSHGOFTAAR, 2019), na qual os autores afirmam que 80% dos trabalhos pesquisados empregam arquiteturas profundas de CNN sobre dados de imagens desbalanceadas. Apesar disso, tais trabalhos podem ser adaptados para outras arquiteturas e outros tipos de problemas. Entretanto, ainda são poucas as pesquisas sobre a interação entre *Deep Learning* e *Big Data* no contexto de bases desbalanceadas (JOHNSON; KHOSHGOFTAAR, 2019).

2.3.2.1 Aprendizado Sensível ao Custo

No Aprendizado Sensível ao Custo (*Cost Sensitive Learning*) o erro de uma classificação é associado a um custo correspondente, minimizando o impacto de uma classificação baseada em acurácia geral (NGUYEN; BOUZERDOUM; PHUNG, 2009; MALDONADO; MONTECINOS, 2014; LÓPEZ et al., 2013). Os valores dos custos podem ser obtidos por meio de um especialista ou outras formas, e.g., meta-aprendizado. Tais valores são armazenados em uma Matriz de Custo.

Na Tabela 2 é apresentado um exemplo de um caso binário. As letras R e P significam Real e Predito, respectivamente. As linhas da matriz indicam a classe real das instâncias, enquanto as colunas indicam a classe predita por algum classificador. Neste exemplo, predizer como 1 uma instância que na verdade é da classe 2, ou, do contrário, predizer como 2 uma instância que é da classe 1, possui um custo associado de 1. A classificação correta, por outro lado, não possui qualquer custo.

Tabela 2 – Matriz de Custo.

(R) / (P)	**Classe 1**	**Classe 2**
Classe 1	0	1
Classe 2	1	0

Quando o Aprendizado Sensível ao Custo é utilizado no contexto de classes desbalanceadas, o erro de classificação de algumas classes pode ter um custo maior. Isto é exemplificado na Tabela 3, onde é possível observar que é mais importante classificar corretamente a classe 2. Situações semelhantes a esta são comuns nas bases de dados do mundo real. Por exemplo, deve ser mais custoso classificar uma pessoa doente como saudável do que classificar uma pessoa saudável como doente. No primeiro caso, o paciente pode vir a óbito por negligência, enquanto no segundo caso é improvável haver sérias consequências.

Tabela 3 – Matriz de Custo com valores diferentes.

(R) / (P)	**Classe 1**	**Classe 2**
Classe 1	0	0.5
Classe 2	1	0

Com a presença de uma matriz de custos, uma instância deve ser classificada de acordo com o menor custo esperado. O custo esperado (ou risco) $R(i|x)$ de classificar uma

instância x como pertencente à classe i pode ser expressado como:

$$R(i|x) = \sum_j P(j|x) \cdot C(i,j) \tag{2.13}$$

Em que $P(j|x)$ é a probabilidade de classificar uma instância x como pertencente à classe j, e $C(i,j)$ é o custo de classificar como j uma instância da classe i. Logo, um classificador só deverá associar a classe i a uma instância x se, e somente se:

$$P(i|x) \cdot C(j,i) + P(j|x) \cdot C(j,j) \leq P(i|x) \cdot C(i,i) + P(j|x) \cdot C(i,j) \tag{2.14}$$

Considerando que $C(i,i) = C(j,j) = 0$, a equação 2.14 torna-se:

$$P(i|x) \cdot C(j,i) \leq P(j|x) \cdot C(i,j) \tag{2.15}$$

Na literatura, as abordagens sensíveis ao custo para classes desbalanceadas podem ser divididas em *abordagem direta* e *meta-aprendizado* (NGUYEN; BOUZERDOUM; PHUNG, 2009; LÓPEZ et al., 2013).

A ideia da abordagem direta é utilizar os custos diretamente nos classificadores. No caso de árvores de decisão os custos podem ser utilizados na escolha do melhor atributo que servirá de nó para uma ramificação (LING et al., 2004). Outro caso é na utilização de algoritmos evolucionários, onde o custo pode ser parte da função de aptidão do modelo (TURNEY, 1995).

O meta-aprendizado transforma algoritmos não sensíveis ao custo em algoritmos sensíveis ao custo. Essa transformação pode acontecer através de um mecanismo de pré-processamento ou de pós-processamento, de forma a deixar o algoritmo original sem modificações. Em outras palavras, o meta-aprendizado entrega ao classificador dados previamente tratados, ou trata as saídas dos classificadores. Um exemplo é associar pesos às instâncias de acordo com a distribuição das classes. Desta forma, instâncias de classes menores são associadas a pesos maiores. Um classificador treina com as instâncias ponderadas, sem sofrer alterações em seu algoritmo.

2.4 MÚLTIPLAS CLASSES DESBALANCEADAS

O problema de classes desbalanceadas é bastante abordado na literatura no caso de bases binárias (FERNÁNDEZ et al., 2013; WANG; YAO, 2012; KOÇO; CAPPONI, 2013; SILVA; ZANCHETTIN, 2016a). Contudo, existem muitas bases de dados com múltiplas classes desbalanceadas, as quais também necessitam atenção. Nessas bases, a distribuição das instâncias pode ser diferente em três ou mais classes distintas.

A presença de várias classes (Figura 7) e, consequentemente, diferentes distribuições ou conceitos, pode elevar a dificuldade do aprendizado no modelo. Ademais, os classificadores

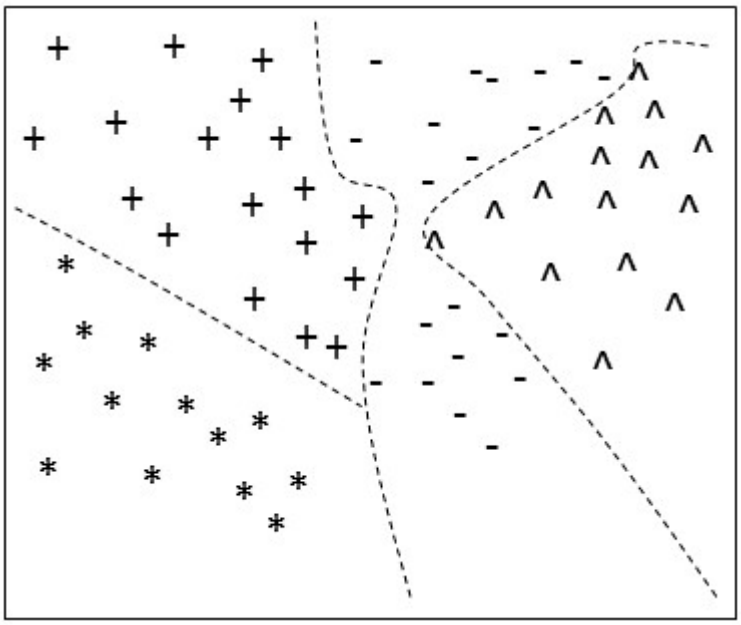

Figura 7 – Exemplo de múltiplas classes desbalanceadas. Observa-se que a separação de uma classe das demais ora é simples, ora é complexa.

podem se comportar de maneira diferente nestes casos em relação ao seu comportamento em bases binárias (SILVA; ZANCHETTIN, 2016b). Pode acontecer de todas as classes de uma base terem bordas entre si, ou somente algumas classes compartilharem suas bordas. Com o desbalanceamento, é possível que somente uma classe seja prejudicada durante o aprendizado. Uma classe pode *dominar* todas as outras, ou seja, um classificador consideraria as outras classes como ruído. Outra possibilidade é que um conjunto de classes domine outro conjunto de classes, ou seja, um classificador conseguirá uma boa generalização somente sobre tal grupo. Em outras palavras, o desafio apresentado pelo desbalanceamento em bases binárias torna-se potencialmente maior em bases com múltiplas classes.

As soluções propostas para as bases binárias podem não ser diretamente aplicáveis neste caso, ou então podem atingir um desempenho abaixo do esperado (FERNÁNDEZ et al., 2013). Um problema com múltiplas classes desbalanceadas pode também requerer um foco diferente. Por exemplo, em um caso binário o foco é classificar corretamente a classe minoritária, mesmo que isso signifique uma certa perda de desempenho para a classe majoritária (FARQUAD; BOSE, 2012). Por outro lado, em uma base de dados com múltiplas classes, pode não haver uma classe *principal*.

Na literatura é possível encontrar três formas de se abordar problemas com múltiplas classes desbalanceadas:

- **Estratégia direta**: consiste em classificadores capazes de lidar com todas as classes de uma vez. Árvores de Decisão ou Redes Neurais Artificiais são exemplos desta abordagem;
- **Estratégia Um-versus-Um (UVU)**: ou um-contra-um. Consiste em dividir o

problema em casos binários, tantos quanto forem as combinações entre pares de classes. Classificadores são treinados em cada par, e as saídas dos classificadores para cada instância de teste são combinadas na decisão final (FERNÁNDEZ et al., 2013; HASTIE; TIBSHIRANI, 1998; GALAR et al., 2011; WANG; YAO, 2012). Na literatura, é comum serem referidos como *One-versus-One* (OVO) ou *One-Against-One* (OAO);

- **Estratégia Um-versus-Todos (UVT)**: ou um-contra-todos. Consiste em treinar um classificador para cada classe, de forma que esta é distinguida das demais classes (FERNÁNDEZ et al., 2013; GALAR et al., 2011; WANG; YAO, 2012). Na literatura é comumente referido como *One-versus-All* (OVA) ou *One-Against-All* (OAA).

As técnicas que transformam problemas de múltiplas classes em problemas binários se tornaram bastante populares (GALAR et al., 2011; FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011). Isto se deu pelo fato de que construir um classificador para apenas duas classes normalmente é mais simples. Tais técnicas — também conhecidas como binarização ou decomposição — possuem em comum a construção de comitês de classificadores. Consequentemente as mesmas compartilham dos benefícios inerentes do uso de comitês no desempenho final da classificação.

Galar et al. (2011) conduzem um estudo comparando UVU e UVT, utilizando várias formas de agregação do comitê de classificadores. Pouco depois Fernández et al. (2013) conduzem outro estudo, acrescentando à comparação algoritmos *ad hoc*. Existem algumas diferenças entre os dois trabalhos citados. O último preferiu focar no pré-processamento dos dados, analisando diversas formas de reamostragem das classes. Ainda, quanto às estratégias UVU e UVT, apenas uma forma de agregação das saídas do comitê de classificadores foi levada em consideração.

Apesar das estratégias de binarização serem populares, as mesmas possuem algumas desvantagens. Por exemplo, com UVU cada classificador é treinado somente para duas classes, as quais necessita distinguir. As instâncias de outras classes continuam desconhecidas. Durante a fase de teste, uma instância desconhecida é apresentada a todos os classificadores. Então, cada classificador fornece a pontuação, ou grau de pertinência, da instância de teste para cada uma de suas duas classes. A classificação final ocorre ao se agregar as pontuações fornecidas por todos os classificadores. Percebe-se que este processo leva em conta tanto os classificadores competentes quanto os não competentes, ou seja, classificadores treinados para uma classe e classificadores treinados para outra classe. Este fator pode ser decisivo para um erro de classificação (GALAR et al., 2013). Logo, o uso, ou a descoberta dos classificadores mais apropriados, acaba se tornando outro problema a ser resolvido.

Nota-se que a tentativa de facilitar o processo de aprendizado e classificação por meio de binarização pode se tornar complexa. Por outro lado, um algoritmo de estratégia direta pode ser menos complexo e obter resultados melhores, uma vez que o treinamento

se dá com todo o conhecimento disponível (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011; OU; MURPHEY, 2007).

Na literatura é possível encontrar algumas soluções, tanto em nível de dados, quanto em nível de algoritmo, que utilizam a estratégia direta. Em nível de dados pode-se citar o *Static* SMOTE e MRBF (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011), MDO (ABDI; HASHEMI, 2016), MDO+ e AMDO (YANG et al., 2018). Em nível de algoritmo pode-se citar o AdaBoost.NC (WANG; YAO, 2012), AdaC2.M1 (SUN; KAMEL; WANG, 2006), AMCS (YIJING et al., 2016) e CoMBo (KOÇO; CAPPONI, 2013). Como exemplo de um algoritmo de decomposição recente na literatura, pode-se citar o DECOC (BI; ZHANG, 2018).

O *Static* SMOTE (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011) é uma adaptação do SMOTE para bases com múltiplas classes. Neste algoritmo, a menor classe da base é selecionada e então passa pelo processo do SMOTE, acrescentando a mesma quantidade de instâncias que a classe possui no conjunto de treinamento original. Este processo é repetido quantas vezes for o valor da quantidade de classes. O conjunto de treinamento modificado é então entregue a um classificador.

O *Memetic Radial Basis Function* (MRBF) é um classificador que utiliza um Algoritmo Memético (MOSCATO; COTTA, 2003) para otimizar uma Rede Neural *Radial Basis Function* (RBF) (FREEMAN; SAAD, 1995). Fernández-Navarro, Hervás-Martínez e Gutiérrez (2011) propõem duas adaptações ao MRBF para bases com múltiplas classes, ou seja, o *Static SMOTE RBF* (SSRBF) e *Dynamic SMOTE RBF* (DSRBF). A primeira das adaptações é bastante similar ao MRBF, com a diferença de utilizar o *Static* SMOTE durante o pré-processamento, para atenuar o desbalanceamento. O DSRBF, entretanto, utiliza o SMOTE no pré-processamento apenas na menor classe, caso o valor de sua probabilidade *a priori* seja menor que o de um limite pré-estabelecido. Em algumas gerações do Algoritmo Memético, o SMOTE é aplicado novamente, porém, somente na classe que estiver gerando o menor desempenho.

O *Mahalanobis Distance-based Over-sampling* (MDO) (ABDI; HASHEMI, 2016) gera instâncias sintéticas para as menores classes levando em consideração o espaço de Componentes Principais (F.R.S., 1901). Os dados sintéticos mantêm a mesma Distância Mahalanobis (MAHALANOBIS, 1936) da média de uma determinada classe. Sua criação segue a variação da classe, preservando a estrutura de covariância. Entretanto, o MDO só consegue lidar com dados numéricos. Para superar esta limitação e lidar com algumas desvantagens do algoritmo, Yang et al. (2018) propuseram o *Adaptive Mahalanobis Distance-based Over-sampling* (AMDO). Os autores estendem ou melhoram o MDO em três aspectos. O primeiro é adaptando a maneira de se lidar com os dados, agora sendo possível lidar com dados nominais ou mistos; o aspecto seguinte é a redução do risco de geração exagerada de instâncias sintéticas e, por fim, uma nova estratégia para geração de exemplos para evitar a criação de instâncias não realistas, ou seja, sem um componente imaginário. Além

do AMDO, o último aspecto é aplicado diretamente ao MDO, o que levou os autores a distinguir essa variação como MDO+.

O algoritmo AdaC2.M1 (SUN; KAMEL; WANG, 2006) é uma adatapção do AdaC2 (SUN; WONG; WANG, 2005) para bases com múltiplas classes. Ambos são uma variação do AdaBoost (FREUND; SCHAPIRE, 1996). O AdaC2.M1 é um algoritmo sensível ao custo, cujos valores ótimos para cada classe são encontrados através de um Algoritmo Genético (AG) (HOLLAND, 1992), levando em conta o desbalanceamento. Após encontrar a melhor configuração de custos, os mesmos são utilizados na fase de *boosting*. Entretanto, devido ao custo de tempo do AG, Wang e Yao (2012) propuseram o AdaBoost.NC, o qual utiliza Aprendizado de Correlação Negativa (LIU; YAO, 1999; ISLAM et al., 2008) e enfatiza a diversidade do comitê de classificadores durante o treinamento. A diferença do AdaBoost.NC está na atualização dos pesos das instâncias, que depende das saídas dos classificadores da iteração corrente e de todo o comitê de classificadores. Além disso, os pesos iniciais são assinalados em proporção inversa ao tamanho das classes.

O *Adaptive Multiple Classifier System* (AMCS) (YIJING et al., 2016) é um algoritmo relativamente complexo, que pré-processa os dados e organiza comitês de classificadores adaptativamente de acordo com o tipo de base de dados. O algoritmo inicia ao escolher o esquema de comitê de classificadores, que pode ser o AdaBoost.M1, *Under-Sampling Balanced Ensemble* (USBE) (SUN et al., 2015) ou *Over-Sampling Balanced Ensemble* (OSBE) (CHAWLA et al., 2003). A partir de então o conjunto de treinamento está dividido em vários subgrupos. Os subgrupos passam por um processo de Seleção de Atributos (SA), dentre dois possíveis, e então são repassados a algum classificador, dentre cinco possíveis. Por fim, os resultados de cada subgrupo são agregados através de alguma regra que, dependendo do esquema de comitê de classificadores, pode ser uma entre cinco possibilidades. Em cada passo, a escolha dentre as alternativas é feita de acordo com o tipo de base de dados. Os autores combinaram a taxa de desbalanceamento (IR), a quantidade de atributos e a quantidade de classes para criar oito tipos diferentes de bases de dados.

O último dos classificadores citados que utiliza a estratégia direta é o *Confusion Matrix Boosting* (CoMBo) (KOÇO; CAPPONI, 2013). Este algoritmo é uma extensão do AdaBoost.MM (MUKHERJEE; SCHAPIRE, 2013), que minimiza a norma empírica da matriz de confusão de forma gananciosa, ou *greedy*. Este algoritmo também utiliza uma matriz de custo, a qual não é utilizada *a priori* no processo de aprendizado. Sua atualização ocorre ao fim de cada iteração onde os custos refletem a dificuldade de classificar corretamente uma determinada instância. A regra de atualização da matriz foi adaptada para considerar o tamanho das classes.

O *Diversified Error Correcting Output Codes* (DECOC) (BI; ZHANG, 2018) é um exemplo de algoritmo *estado-da-arte* que utiliza a estratégia de decomposição para múltiplas classes desbalanceadas. É uma combinação entre os algoritmos *Diversified One-Versus-One* (DOVO) (KANG; CHO; KANG, 2015) e ECOC (DIETTERICH; BAKIRI, 1991). O DOVO

tem por objetivo encontrar o melhor classificador para cada subproblema criado pelo OVO. O ECOC cria um código, ou *codeword*, para cada classe a fim de obter a maior distância entre várias classes. Cada classificador treina sobre as combinações de códigos, ou seja, entre combinações de classes. Levando em conta que cada classificador contribui de forma diferente para a predição final, o DECOC assinala pesos de forma a favorecer as menores classes.

As soluções para múltiplas classes desbalanceadas precisam de métricas apropriadas para o cálculo de seus desempenhos, da mesma forma que as soluções para bases binárias. Entretanto, para este novo caso ainda é necessária outra adaptação das métricas de avaliação. Por exemplo, as duas métricas mais utilizadas no problema de classes desbalanceadas são AUC e *G-mean*. Essas duas métricas, entretanto, só podem ser usadas em casos binários.

Uma extensão da métrica AUC para múltiplas classes foi proposta por Hand e Till (2001):

$$MAUC = \frac{2}{C(C-1)} \sum_{i<j}^{C} AUC(i,j) \tag{2.16}$$

no qual C é o número de classes. O mesmo valor de desempenho do MAUC por ser calculado usando a média aritmética dos AUCs obtidos em cada par de classes e, por isso, esta métrica é também referida como *AvgAUC*.

Existem duas formas de se calcular a métrica *G-mean* no cenário de múltiplas classes. A mais comum na literatura, também conhecida como *Macro Average Geometric* (MAvG) (SÁNCHEZ-CRISOSTOMO et al., 2014) é definida na seguinte equação:

$$G\text{-}mean = (\prod_{i=1}^{C} TVP_i)^{1/C} \tag{2.17}$$

no qual C é a quantidade de classes e TVP_i é a taxa de acerto, ou verdadeiro positivo, da classe i. A outra forma é semelhante ao *AvgAUC*, no qual o valor final é a média aritmética dos valores de *G-mean* de cada par de classes.

Outra métrica que pode ser estendida de forma similar é a Acurácia Balanceada (2.9), também referida na literatura como *Macro Average Arithmetic* (MAvA) (SÁNCHEZ-CRISOSTOMO et al., 2014):

$$\text{Acurácia Balanceada} = \frac{1}{C} \sum_{i=1}^{C} \text{TVP}_i \tag{2.18}$$

Outra vez C é quantidade de classes presentes na base de dados e TVP_i é a taxa de acerto da classe i. Contudo, diferentemente do caso binário, em que a Acurácia Balanceada é equivalente ao AUC, no caso de múltiplas classes nem sempre essa equivalência será verdadeira (SILVA; ZANCHETTIN, 2016b). Ainda, com a presença de uma classe extremamente rara na base de dados, o MAUC é preferível.

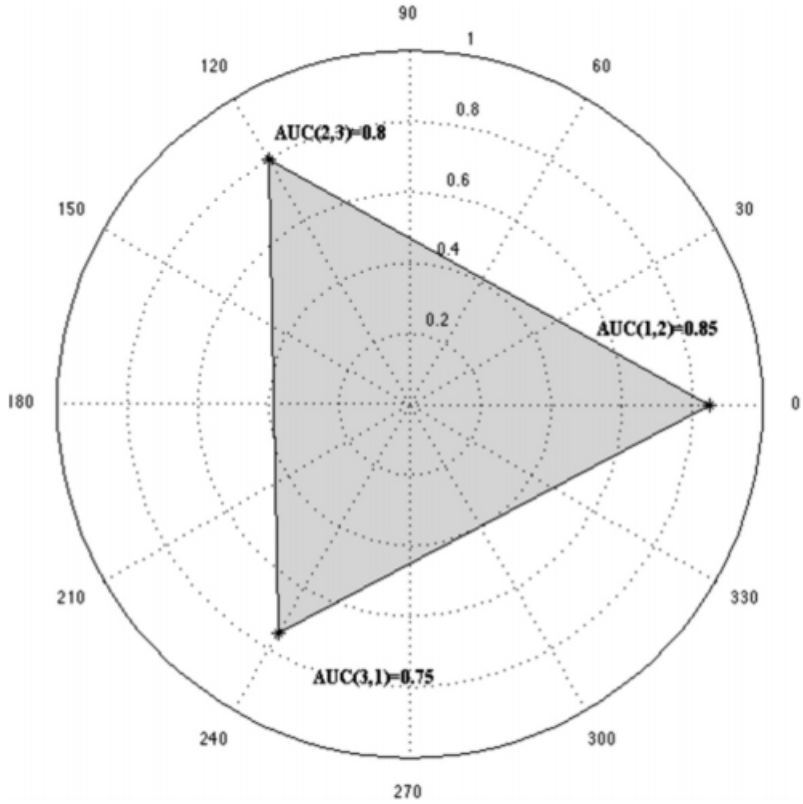

Figura 8 – Exemplo de diagrama polar com AUCs para três classes (fonte: (YIJING et al., 2016)).

O cálculo do MAUC ou *AvgAUC* possui um pequeno defeito (YIJING et al., 2016). Diferentes classificadores podem obter AUCs diferentes em algumas combinações de classes, entretanto, o valor final pode ser o mesmo, o que dificulta saber se algum classificador foi de fato melhor ao classificar as menores classes, por exemplo. Uma maneira de se contornar tal defeito é com uma extensão chamada AUCarea (HASSAN et al., 2010), a qual marca todos os valores de AUC em uma coordenada polar. A área coberta pelo diagrama polar corresponde ao desempenho final obtido com esta métrica, portanto, quanto maior a área, melhor o desempenho. A Figura 8 apresenta um exemplo de diagrama polar, com AUCs para três classes marcadas, e sua respectiva área em evidência.

AUCarea pode ser calculada da seguinte forma:

$$AUCarea = \frac{(\sum_{i=1}^{q-1} r_i \times r_{i+1}) + (r_q \times r_1)}{q} \tag{2.19}$$

na qual $q = C_C^2$, ou seja, a quantidade de combinações dois em dois do número de classes, e r_i é o *i-ésimo* AUC. Esta fórmula é uma adaptação feita pelos autores em (YIJING et al., 2016) para que os valores fiquem normalizados no intervalo [0, 1]. Ainda, é importante ressaltar que esta métrica é mais sensível a baixos valores de AUC que o MAUC.

Por fim, outras métricas utilizadas em soluções para múltiplas classes desbalanceadas

são a Mínima Sensibilidade (MS), *Cohen's kappa* (COHEN, 1960; GALAR et al., 2011) e *Class Balance Accuracy* (CBA) (MOSLEY, 2013). A MS é definida como a menor taxa de acerto dentre todas as classes. Formalmente:

$$MS = min\{S_i; i = 1, ..., C\} \tag{2.20}$$

na qual C é quantidade de classes e S_i é a taxa de acerto da classe i.

As métricas *kappa* e CBA são construídas a partir da matriz de confusão para múltiplas classes (Tabela 4).

Tabela 4 – Matriz de Confusão para Múltpilas Classes.

	Predito C_1	**Predito** C_2	...	**Predito** C_m	**Total**
C_1 **Real**	h_{11}	h_{12}	...	h_{1m}	T_{l1}
C_2 **Real**	h_{21}	h_{22}	...	h_{2m}	T_{l2}
$\vdots$			$\ddots$		
C_m **Real**	h_{m1}	h_{m2}	...	h_{mm}	T_{lm}
Total	T_{c1}	T_{c2}	...	T_{cm}	T

A métrica *kappa* avalia a quantidade de acertos que podem ser atribuídas ao classificador, relativo a todas as classificações que não podem ser atribuídas à *sorte*. Os valores podem variar entre -1 (desacordo total), 0 (classificação aleatória) e 1 (concordância perfeita). Em outras palavas, a métrica *kappa* calcula o desempenho de um classificador compensando acertos aleatórios (GALAR et al., 2011).

$$kappa = \frac{n \sum_{i=1}^{m} h_{ii} - \sum_{i=1}^{m} T_{li} T_{ci}}{n^2 - \sum_{i=1}^{m} T_{li} T_{ci}} \tag{2.21}$$

onde h_{ii} é a taxa de acerto para cada classe, n é o número de instâncias, m é a quantidade de classes, $T_{li} = \sum_{j=1}^{m} h_{ij}$ e $T_{ci} = \sum_{j=1}^{m} h_{ji}$.

O CBA procura balancear a Precisão e a Sensibilidade de cada classe. Quando há um desbalanceamento entre Precisão e Sensibilidade um processo conservador é empregado para que a menor das duas medidas seja a representante da classe em questão (MOSLEY, 2013).

$$CBA = \frac{\sum_{i}^{k} \frac{h_{ii}}{max(T_{li}, T_{ci})}}{k} \tag{2.22}$$

onde k é o número de classes, h_{ii} é a taxa de acerto para cada classe, $T_{li} = \sum_{j=1}^{m} h_{ij}$ e $T_{ci} = \sum_{j=1}^{m} h_{ji}$.

A métrica sobre a base também se faz presente no contexto de múltiplas classes desbalanceadas. Até então, praticamente todos os estudos ainda utilizam a taxa IR para calcular o desbalanceamento de uma base que possui três ou mais classes. A taxa IR, quando aplicada a múltiplas classes desbalanceadas, exclui informações sobre as demais classes. Para lidar com essa questão podem ser encontradas na literatura duas novas taxas: *Imbalance Degree* (ID) (ORTIGOSA-HERNáNDEZ; INZA; LOZANO, 2017) e *Multiclass Imbalance Ratio* (MIR) (SILVA; ZANCHETTIN, 2016b).

A taxa ID foi construída para possuir três propriedades: (1) ser um valor único real inteligível na faixa [0, C), sendo C a quantidade de classes; (2) ser instanciável por qualquer métrica ou nível de dissimilaridade e (3) ser uma função injetiva para distribuições de classes diferentes (ORTIGOSA-HERNáNDEZ; INZA; LOZANO, 2017). Formalmente esta taxa pode ser calculada como segue:

$$ID(\zeta) = \frac{d_\Delta(\zeta, \mathbf{e})}{d_\Delta(\iota_m, \mathbf{e})} + (m - 1) \tag{2.23}$$

na qual ζ é a distribuição da base, ou seja, as probabilidades ou tamanhos das classes; $\mathbf{e}$ é a probabilidade ou tamanho das classes em um contexto balanceado, ou seja, para n instâncias e c classes, $\mathbf{e} = n/c$; m é a quantidade de classes minoritárias; d_Δ é a função de similaridade ou distância escolhida; ι_m é a distribuição mostrando exatamente m classes minoritárias com a maior distância a $\mathbf{e}$.

O estudo sobre o ID (ORTIGOSA-HERNáNDEZ; INZA; LOZANO, 2017) revelou que esta taxa é mais sensível em expressar o desbalanceamento em relação ao IR. Além disso os autores sugerem que as distâncias Hellinger ou Manhattan são recomendadas como função de distância ou similaridade.

Apesar de ser uma contribuição ao campo de múltiplas classes desbalanceadas, a taxa ID apresenta alguns problemas. O primeiro é não ser simples de se calcular, exigindo um certo conhecimento estatístico. A segunda dificuldade é a presença do parâmetro de função de similaridade ou distância. Valores diferentes para este parâmetro resultam em valores diferentes para o ID, o que pode causar alguma confusão ou dificuldade de entendimento ou interpretação. Por fim, existe a dificuldade de como definir quais classes são minoritárias. A partir do estudo que apresenta o ID, uma classe minoritária seria aquela cuja probabilidade *a priori* seja menor que uma probabilidade balanceada, ou seja, $1/C$, sendo C a quantidade de classes. Por exemplo, em uma base cujas classes tenham 500, 300 e 100 instâncias, as duas primeiras classes seriam consideradas majoritárias. Contudo, se a última classe tiver 200 instâncias, a classe do meio passa a ser minoritária. Portanto, apesar da definição dada pelo estudo, quando considerar uma classe como minoritária ainda é uma questão em aberto.

Em contrapartida, a taxa MIR é mais simples e próxima do conceito da taxa IR. Ela expressa o quão heterogêneo são os tamanhos das classes de uma base de dados. Uma base de dados balanceada terá um MIR com valor zero. Por outro lado, quanto

maior for este valor, maior é a diferença de tamanho entre as classes. Logo, uma base de dados com um valor alto de MIR provavelmente sofrerá influências negativas advindas do desbalanceamento.

Considerando-se $C = \{c_1, c_2, ..., c_n\}$ como o conjunto de classes e x_c o número de instâncias da classe c_n, a taxa MIR pode ser definida como:

$$MIR = \sum_{c=1}^{n} \frac{\frac{\sum_{c=1}^{n} x_c}{n}}{x_c} - n \tag{2.24}$$

O numerador $(\sum_{c=1}^{n} x_c/n)$ calcula a quantidade de instâncias que cada classe deveria ter para que a base fosse balanceada. O denominador, como definido anteriormente, é o tamanho de cada classe. O resultado da divisão é uma taxa de desbalanceamento. A soma de todas as taxas e posterior subtração do número de classes compõem o valor de MIR.

O último ponto que merece destaque no problema de múltiplas classes desbalanceadas é a questão de validação de experimentos. O método de validação mais comum é a validação cruzada ou *cross-validation* (CV) (COHEN, 1995). De suas variações, as mais usadas são *10-fold* e *hold-out* 33% (SILVA; ZANCHETTIN, 2016a).

Entretanto, devido a uma constante presença de classes raras nas bases desbalanceadas, muitos autores têm preferido utilizar o *5-fold*. Contudo, com a presença de classes extremamente raras, ou seja, com menos de cinco instâncias, o método *hold-out* 33% repetido pode ser uma melhor alternativa, como sugerido em (SILVA; ZANCHETTIN, 2016a). A vantagem do *hold-out* nesta situação é a garantia de que todas as classes estarão presentes tanto no conjunto de treinamento quanto no de teste, exceto se a classe possuir apenas uma única instância.

2.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi introduzido e analisado o problema de bases desbalanceadas. Nas primeiras seções o problema foi explorado sob a perspectiva de bases binárias. Três grandes linhas de pesquisa foram abordadas, a saber, os estudos sobre os efeitos do desbalanceamento, as métricas utilizadas para qualificar os resultados dos classificadores, e formas diferentes de se lidar com o problema.

Quanto à primeira linha de pesquisa, foi possível compreender, por exemplo, que a complexidade da disposição dos dados aliado ao tamanho do conjunto de treinamento posuem bastante influência sobre o problema. Além destes, outros dois fatores foram também pontuados. Nominalmente, casos de classes extremamente raras e o problema de *data set shift*.

A seção que trata da segunda linha de pesquisa contem a explicação de o porquê a Acurácia não ser considerada uma métrica apropriada para ser utilizada no problema de

bases desbalanceadas. Posteriormente foram exploradas diversas métricas que são utilizadas pelos pesquisadores desta área. As principais são: Acurácia Balanceada, AUC e *G-mean*. Além das métricas de desempenho, foi também apresentada a métrica IR, uma métrica sobre o desbalanceamento de bases binárias.

Sobre a terceira linha de pesquisa, as abordagens foram divididas em dois grupos principais, i.e., Abordagem em Nível de Dados e Abordagem em Nível de Algoritmo. Os detalhes sobre cada abordagem, em conjunto com exemplos de algoritmos existentes na literatura, foram também apresentados.

Por fim, o problema de múltiplas classes desbalanceadas foi apresentado. Um dos focos na respectiva seção foi apresentar diferenças em relação ao caso de bases binárias, e como a presença de múltiplas classes aumenta o desafio. As estratégias para se lidar com três ou mais classes foram também abordadas, seguidas de exemplos da literatura. A seção continuou com as métricas de desempenho e desbalanceamento que são apropriadas para o problema, as quais parte são extensões de métricas utilizadas no caso binário. A seção finaliza com um rápido comentário sobre a validação dos experimentos, principalmente na presença de classes extremamente raras.

3 MÉTODO PROPOSTO

Neste capítulo são detalhados o processo de pesquisa e desenvolvimento conduzidos nesta tese. A pesquisa foi iniciada a partir de uma hipótese (Seção 3.1) cuja ideia foi implementada e aprimorada ao longo do doutorado. O aprimoramento do método proposto se deu com a implementação de algumas modificações nos algoritmos principais. Algumas das modificações foram motivadas a partir do próprio algoritmo, enquanto que outras modificações têm suas motivações a partir da literatura.

O desenvolvimento do método proposto pode ser dividido em duas grandes fases (Seções 3.2 e 3.3), representados pelos algoritmos *Voronoi Diagram Based Classifier* (VDBC) e *Dynamic Centroid Insertion and Adjustment* (DCIA). O primeiro algoritmo foi desenvolvido a partir das ideias levantadas durante a formulação da hipótese de pesquisa. O segundo — e principal — algoritmo pode ser visto como evolução do primeiro.

A apresentação de experimentos durante a descrição da proposta dos modelos não é muito usual. Apesar disso, devido a grande quantidade de variações, análises realizadas e evoluções incrementais, neste capítulo, são apresentados alguns dos resultados obtidos na evolução do modelo e que ajudam a justificar algumas das decisões tomadas durante a pesquisa realizada.

A execução de ambos os algoritmos, como também de suas modificações, foi sempre precedida de uma divisão dos dados em um conjunto de treinamento e um conjunto de teste seguindo o modelo *5-fold*. O motivo se deve ao fato de que essa divisão é bastante comum na literatura de tratamento de bases de dados desbalanceados devido às classes raras.

Além das execuções dos algoritmos existem também comparações estatísticas. Nos casos em que os resultados de três ou mais algoritmos são comparados, os testes estatísticos foram feitos com ANOVA (ARMSTRONG; S.V.SLADE; F.EPERJESI, 2000) ou Kruskall-Wallis (KRUSKAL; WALLIS, 1952; KRUSKAL-WALLIS..., 2008). Ambos os testes comparam a significância de diferenças entre três ou mais grupos de amostras independentes, ou seja, analisam a hipótese de que k amostras tenham sido extraídas da mesma população ou de populações idênticas. O teste ANOVA é paramétrico e utilizado neste trabalho quando os resultados dos algoritmos possuem uma distribuição normal. Do contrário, ou seja, quando ao menos um dos resultados não possui uma distribuição normal, o teste Kruskall-Wallis é utilizado por ser não-paramétrico.

Contudo, quando apenas dois resultados são comparados, os testes foram conduzidos com *Student's t-test* (SNEDECOR; COCHRAN, 1980) ou *ranksum* de Wilcoxon (WILCOXON, 1945). Estes testes comparam a significância de diferenças entre dois grupos de amostras independentes, analisando a hipótese de que as duas amostras tenham sido extraídas da mesma população ou de populações idênticas. O *t-test* é paramétrico, portanto, é utilizado

neste trabalho quando os resultados de um algoritmo fazem parte de uma distribuição normal. Por outro lado, o *ranksum* é um teste não-paramétrico e utilizado quando ao menos um dos resultados não possui distribuição normal.

É importante frisar que todos os algoritmos desenvolvidos fazem parte da abordagem em nível de dados baseado na estratégia direta. Ou seja, tanto o VDBC quanto o DCIA e todas as suas modificações pré-processam um conjunto de treinamento. O conjunto de protótipos resultantes é utilizado pelo 1NN para classificação do conjunto de testes. Como o objetivo da tese é trabalhar a nível de dados, se escolheu um dos classificadores mais simples (1NN ou Vizinho mais próximo) para servir de base na análise do desempenho do algoritmo de reamostragem proposto.

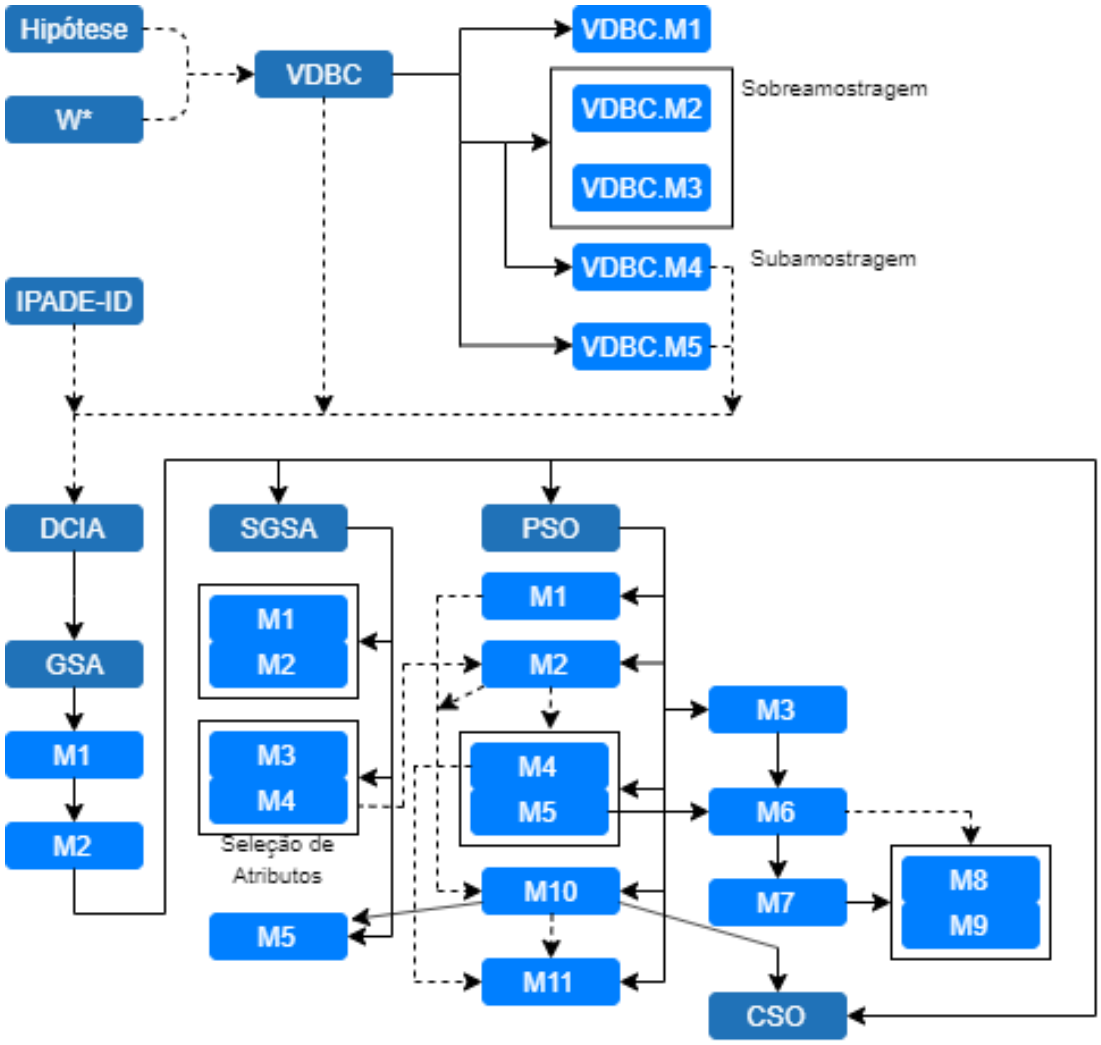

Figura 9 – Fluxograma da evolução do desenvolvimento do trabalho e como os algoritmos desenvolvidos estão relacionados. Uma linha tracejada significa que um algoritmo inflenciou o outro. Uma linha contínua indica que o algoritmo subsequente é modificação direta do algoritmo anterior.

A quantidade de modificações e testes realizados sobre os algoritmos propostos pode deixar o leitor confuso. Portanto, a Figura 9 apresenta um fluxograma explicando a sequên-

cia do desenvolvimento do trabalho e como os algoritmos e suas modificações estão relacionados.

De forma resumida o desenvolvimento deste trabalho se deu da seguinte forma. De início foi formulada uma hipótese, a qual é abordada na Seção 3.1. De forma conjunta com o observado no algoritmo W* (CHANG, 1974) foi possível desenvolver o modelo de classificação VDBC (Seção 3.2). Sobre o VDBC foram testadas cinco modificações, cujos detalhes e resultados são apresentados no Apêndice A. Com as análises dos resultados da quarta e quinta modificações, aliado a influências do algoritmo IPADE-ID (LÓPEZ et al., 2014), foi implementado o algoritmo DCIA (Seção 3.3).

O DCIA pode ser executado utilizando qualquer algoritmo de busca. O primeiro algoritmo de busca utilizado em conjunto com o método foi o *Gravity Search Algorithm* (GSA) (RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009). Sobre essa combinação foram testadas duas modificações, na qual a primeira é alteração direta do *DCIA with Gravity Search Algorithm* (DCIAGSA), enquanto a segunda é alteração direta da primeira modificação, como exemplificado na Figura 9.

Após o DCIAGSA, foram testadas também outras três combinações do DCIA com algoritmos de busca. A segunda combinação envolveu uma versão do GSA, chamada *Simple Gravity Search Algorithm* (SGSA). Sobre esta combinação foram testadas, de início, quatro modificações, as quais as duas primeiras e as duas últimas são mais relacionadas entre si.

Em seguida foram implementados o *DCIA with Particle Swarm Optimization* (DCIAPSO), utilizando o algoritmo de busca PSO, como também modificações sobre este algoritmo. Como mostrado na Figura 9, a segunda modificação foi inspirada pela seleção de atributos testada com o DCIASGSA. Por sua vez, o DCIAPSO.M2 serviu como inspiração para outras modificações. Outro exemplo curioso é o DCIAPSO.M6, o qual pode ser considerado como uma junção das modificações M3 e M5. O DCIAPSO.M10 é uma alteração direta no DCIAPSO, porém construído com inspirações das duas primeiras modificações. A partir de seu sucesso, o mesmo foi utilizado como base para o DCIASGSA.M5 e o *DCIA with Competitive Swarm Optimizer* (DCIACSO).

No decorrer deste capítulo, os algoritmos principais, i.e., DCIAGSA, DCIASGSA, DCIAPSO e DCIACSO são apresentados de forma detalhada. As respectivas modificações de cada algoritmo principal são também descritas, porém seus detalhes e resultados são apresentados no Apêndice B.

A organização do capítulo segue a sequência de desenvolvimento apresentada na Figura 9. O capítulo é então finalizado com um apanhado geral sobre toda a pesquisa e desenvolvimento realizados.

3.1 REVISITANDO A HIPÓTESE DE PESQUISA

A primeira ideia neste trabalho consiste em dividir todo um espaço amostral em várias pequenas células. Para cada célula, uma determinada classe é associada a ela. A partir de então, qualquer instância encontrada em uma célula seria classificada de acordo com a classe da mesma. O objetivo é tentar detalhar, o máximo possível, as regiões de cada classe presente na base de dados amostral.

Alguns dos algoritmos de classificação mais utilizados, como MPLs e SVMs, procuram construir uma função que descreva uma fronteira de separação entre as classes ou um hiperplano de separação no hiperespaço do problema. Ao mesmo tempo, a divisão do espaço amostral em regiões excludentes é uma alternativa simples, e que pode atingir um bom desempenho de classificação. Uma vez que a classificação em bases de dados com múltiplas classes desbalanceadas é complexa, torna-se interessante a análise de uma abordagem simples.

O detalhamento das regiões das classes, e suas diferenças em relação a uma fronteira entre classes, são explicadas a seguir. Para um melhor entendimento, a explicação será acompanhada de figuras sobre uma base de dados desbalanceada sintética (Figura 10), usada como exemplo. Nesta base de dados as classes são dispostas em duas espirais.

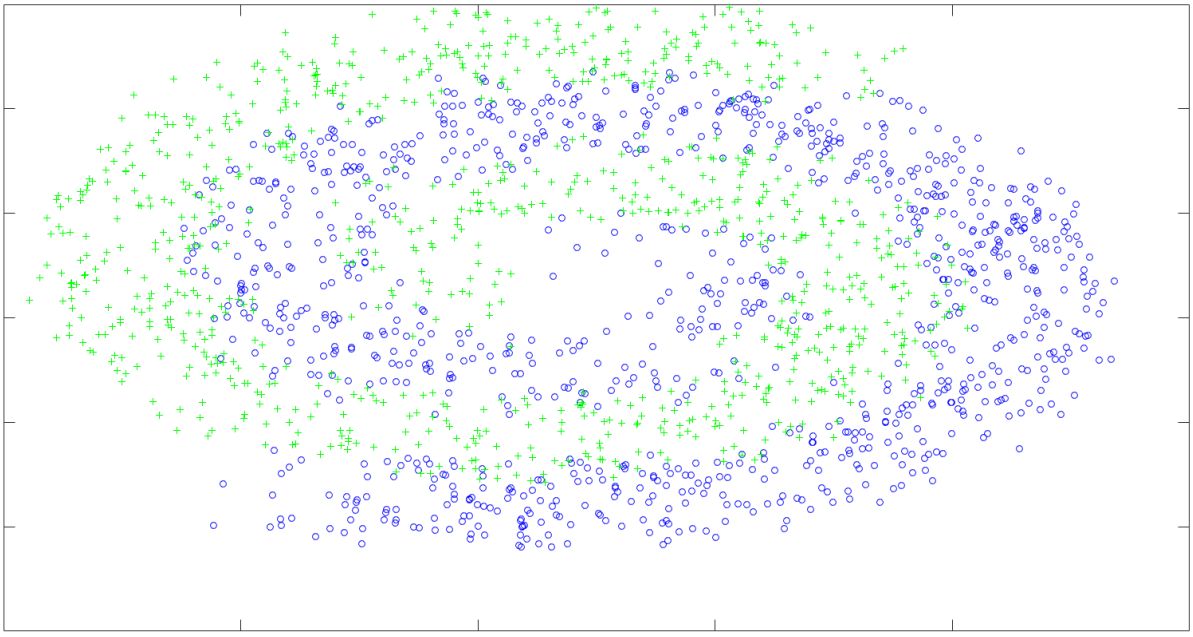

Figura 10 – Exemplo de base de dados desbalanceada em espiral. A classe representada por cruzes verdes é a classe minoritária.

Qualquer algoritmo de classificação que descreva uma fronteira em espiral, pode conseguir um desempenho de classificação razoável neste caso (Figura 11). Entretanto, se as espirais desenhadas puderem seguir um traçado mais complexo (Figura 12), provavelmente o desempenho de classificação irá melhorar, pois várias das instâncias que estão

nas bordas das classes serão apropriadamente classificadas. Contudo, a melhora do desempenho tem como custo o aumento da complexidade da função que gera a fronteira de separação entre as classes.

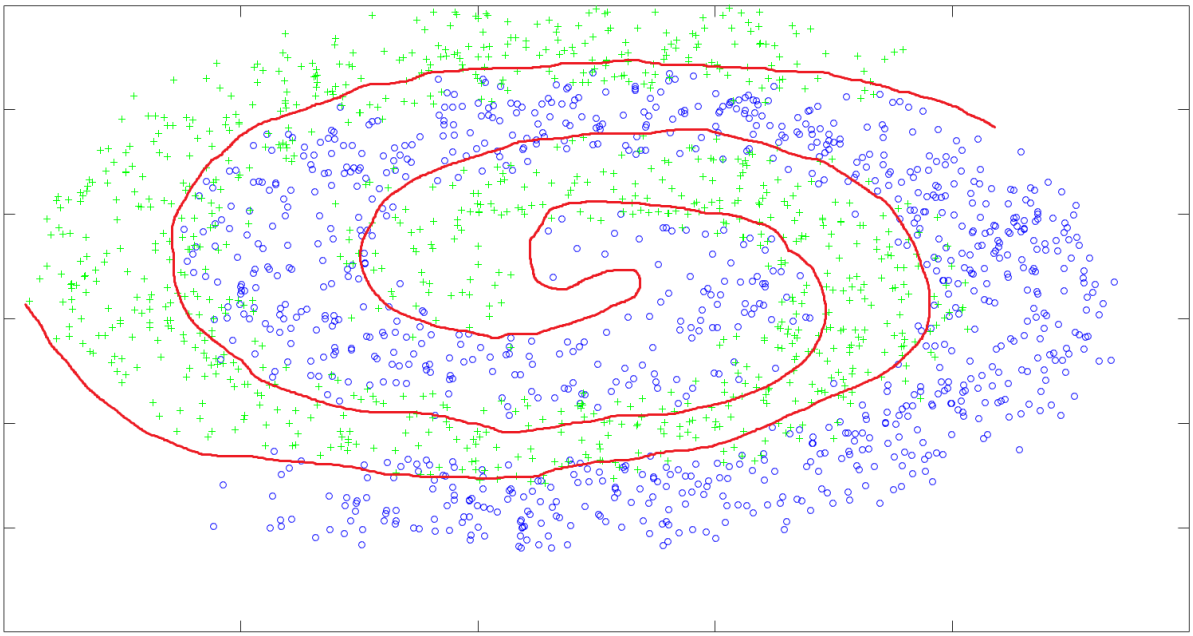

Figura 11 – Base de dados em espiral com uma possível função de separação.

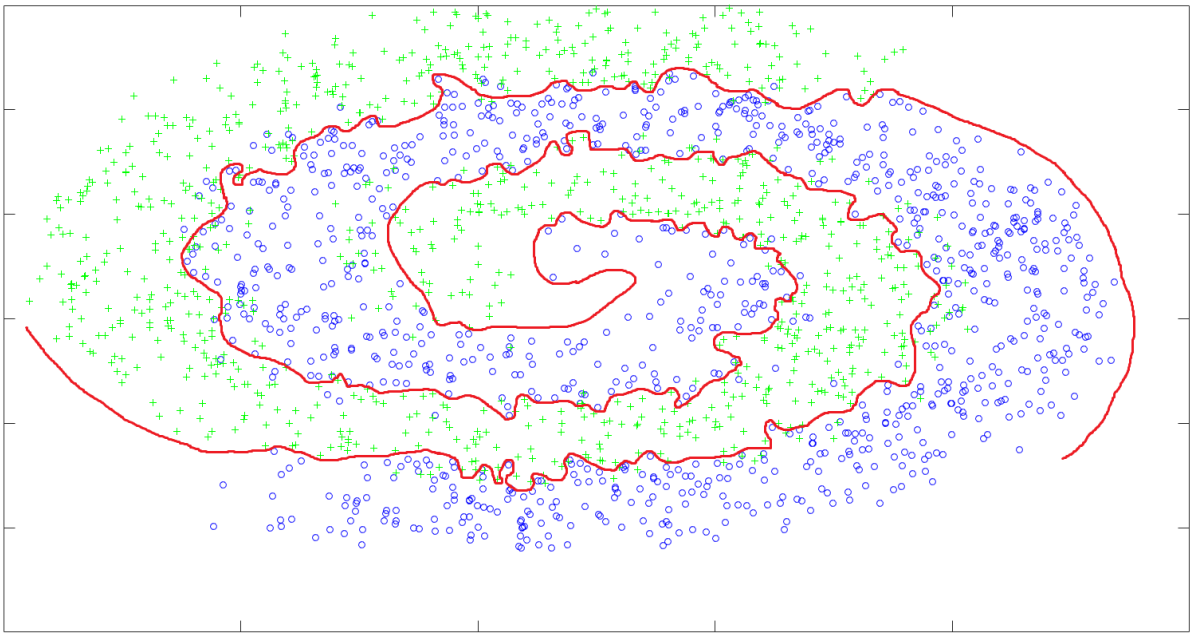

Figura 12 – Base de dados em espiral com uma possível função de separação mais complexa.

Ao se dividir o espaço amostral em uma grade de regiões é possível aumentar a complexidade de separação entre as classes de uma maneira simples. A Figura 13 mostra a

base de dados em espiral junto com uma grade de centroides. Cada centroide da figura poderá ser associado à classe da instância mais próxima. Por consequência, a região em volta de cada centroide passa a ser associada a uma classe específica. A Figura 14 mostra a base dividida em uma série de regiões, onde cada região corresponde a um centroide.

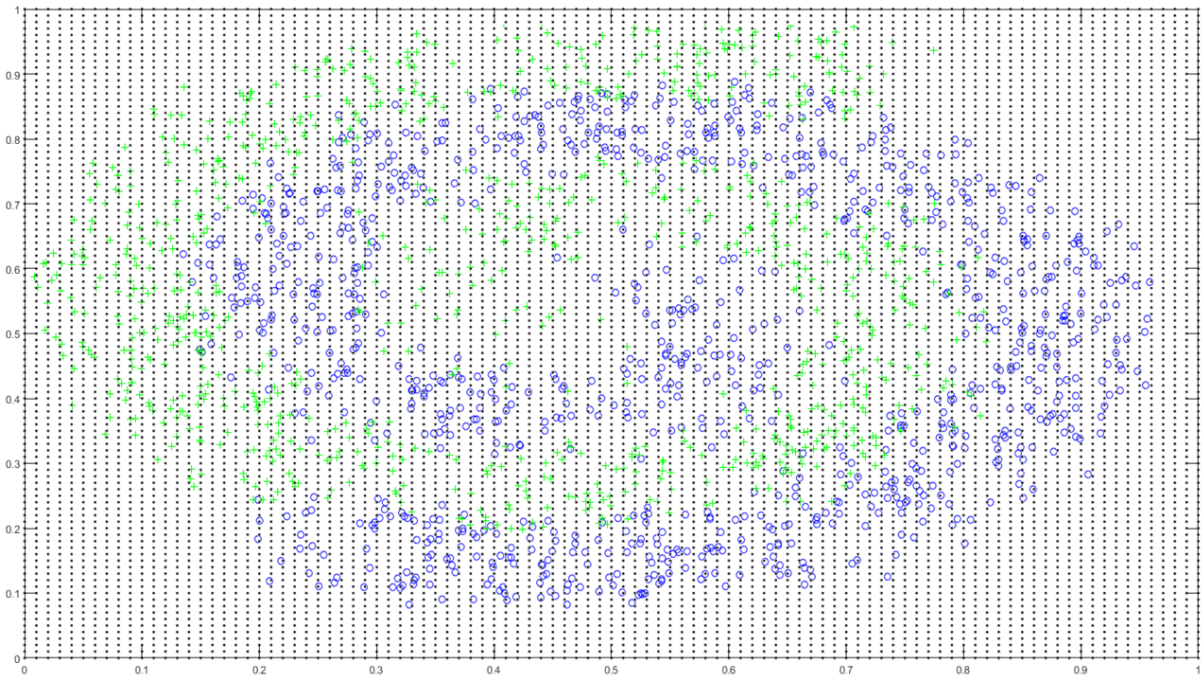

Figura 13 – Base de dados com uma grade de centroides.

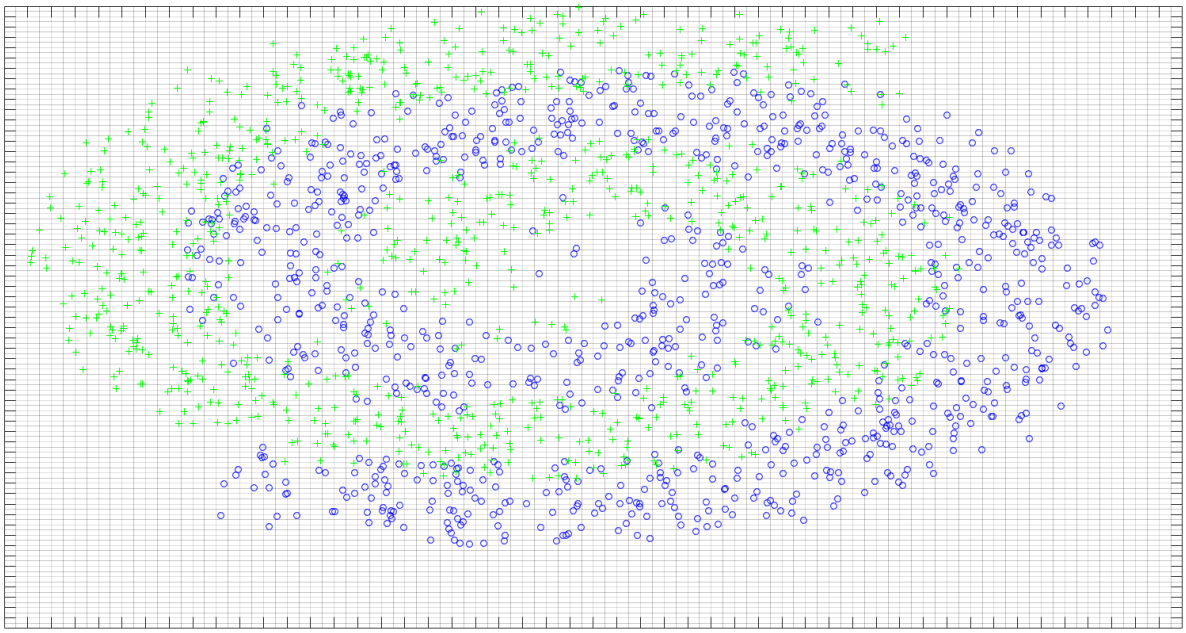

Figura 14 – Base de dados dividida em regiões.

Entretanto ao se observar a Figura 14 é possível notar que algumas regiões possuem instâncias de ambas as classes. Isto significa que durante o teste as instâncias que estejam

nessas regiões poderão ser classificadas incorretamente. Uma solução para isso é aumentar a quantidade de centroides (Figura 15), ou, em outras palavras, dividir o espaço amostral em um número maior de regiões. Isto possibilita uma classificação ainda mais complexa, porém com um desempenho de classificação melhor, uma vez que se torna mais possível cada região conter apenas uma instância.

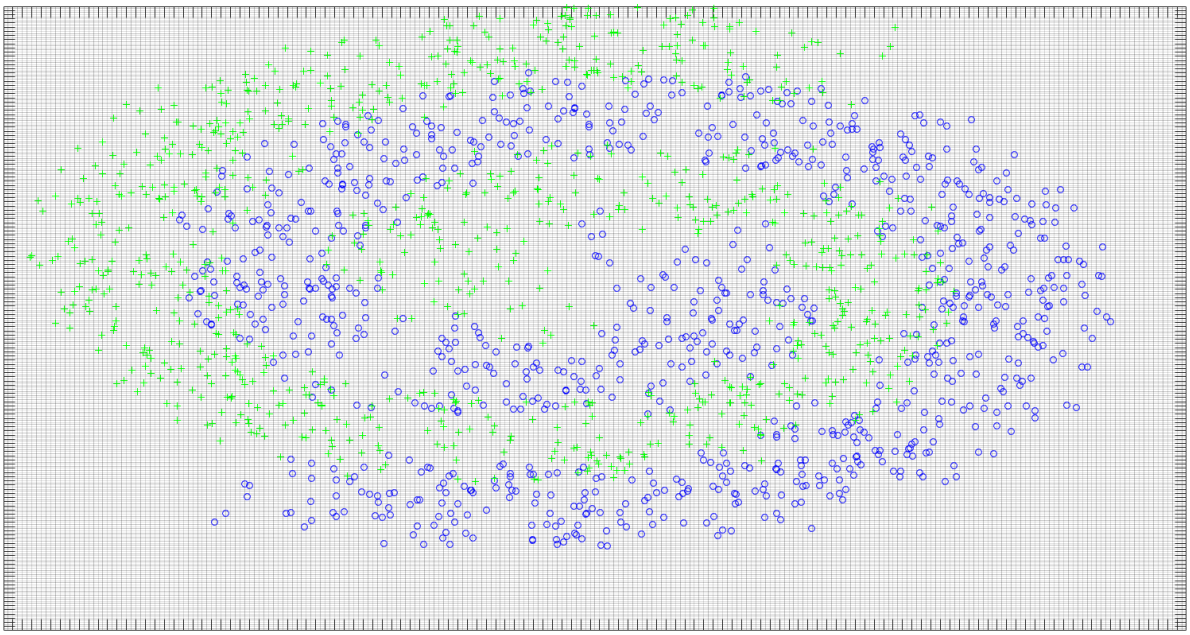

Figura 15 – Base de dados dividida em um número maior de regiões.

Contudo, ao mesmo tempo em que a solução é simples, há um enorme aumento de custo computacional. Levando-se em consideração n valores para cada dimensão d do espaço amostral, o número de centroides será n^d. Para melhor entendimento, considere $n = 10$ e $d = 12$. Neste exemplo, serão dez trilhões (10^{12}) de centroides. Logo, dependendo da base de dados, essa abordagem se torna impraticável.

Uma maneira de se manter a solução e diminuir o custo computacional é utilizando protótipos para representar as regiões de separação (CHANG, 1974; OUGIAROGLOU; EVANGELIDIS, 2012). Um protótipo é um centroide que serve como *representante* de várias instâncias. Desta forma, muitas regiões adjacentes de uma mesma classe podem ser *fundidas* e representadas por apenas um protótipo. Regiões mais complexas, como as bordas entre as classes, podem ser divididas em várias regiões, enquanto o restante do espaço amostral passa a ser dividido em regiões maiores, com poucos protótipos.

Um espaço amostral dividido com protótipos pode ser representado graficamente através de um diagrama de Voronoi (AURENHAMMER, 1991). Neste diagrama, um conjunto de pontos $S = \{s_1, s_2, ..., s_n\}$, também conhecidos como sementes, é usado para dividir as regiões. Cada semente s_i domina uma região, i.e., todos os pontos daquela região são mais próximos de s_i do que de qualquer outra semente.

A criação de um diagrama de Voronoi (Figura 16) pode ser realizada de várias formas. Uma delas é através da técnica *k-means* (MACQUEEN, 1967). Outras formas se utilizam do algoritmo *k-Nearest Neighbors* (k-NN) para se obter um diagrama de Voronoi através da seleção ou geração de protótipos. Por exemplo, o *Condensed Nearest Neighbor* (GOWDA; KRISHNA, 1979; BHATIA; VANDANA, 2010), e o W* (CHANG, 1974).

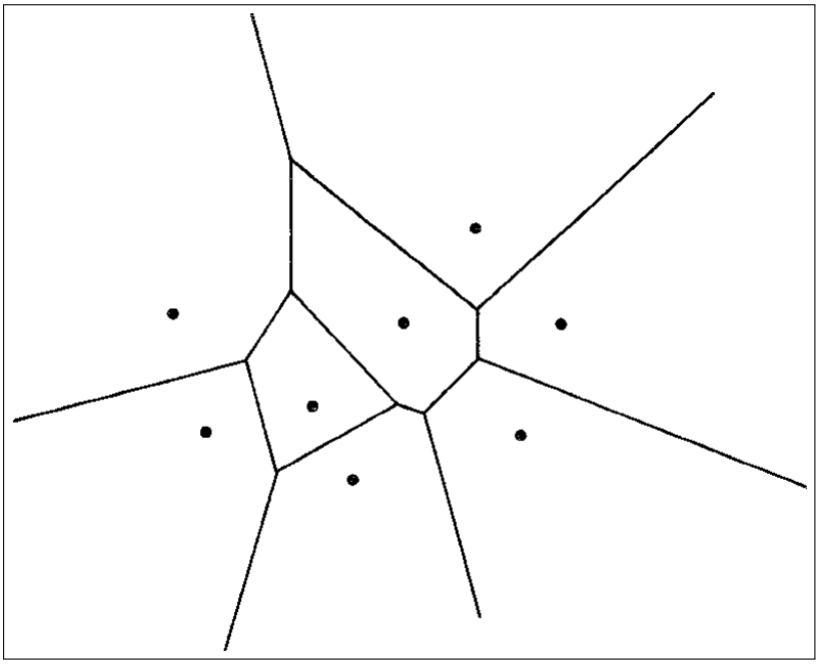

Figura 16 – Exemplo de um diagrama de Voronoi (fonte: (AURENHAMMER, 1991)).

Uma vez que o *k-means* é um algoritmo de agrupamento de dados, o mesmo não gera e não seleciona protótipos. O *Condensed Nearest Neighbor*, contudo, é um algoritmo de Seleção de Protótipos. A ideia de seleção de protótipos é formar um subconjunto de treinamento ao eliminar instâncias menos *significativas*, i.e., instâncias cujos vizinhos sejam da mesma classe (BHATIA; VANDANA, 2010; OUGIAROGLOU; EVANGELIDIS, 2012). Por outro lado, a ideia de geração de protótipos é criar centroides de acordo com a similaridade entre as instâncias. O conjunto de treinamento é substituído por um novo conjunto, contendo os protótipos gerados (BHATIA; VANDANA, 2010).

Dentre os métodos citados para a criação de um diagrama de Voronoi, o mais próximo da hipótese avaliada é a Geração de Protótipos, pois o objetivo primário é utilizar um conjunto de centroides para guiar o processo de classificação. Com a primeira ideia refinada foi possível seguir para o próximo estágio da pesquisa.

O próximo estágio consiste na criação e teste do algoritmo VDBC (*Voronoi Diagram Based Classifier*), ou em uma tradução *Classificador Baseado em Diagrama de Voronoi.*

3.2 *VORONOI DIAGRAM BASED CLASSIFIER*

O processo de redução de instâncias tem por objetivo encontrar o melhor conjunto reduzido que represente o conjunto de treinamento original, com a menor quantidade de instâncias, ou protótipos, possível. O uso de métodos de redução de instâncias pode ser aplicado a bases desbalanceadas como uma solução em nível de dados (LÓPEZ et al., 2014).

Um dos primeiros métodos de redução de instâncias, sendo também um algoritmo de geração de protótipos, é o algoritmo W* (CHANG, 1974). Este algoritmo é bastante simples, porém eficiente. Seu funcionamento é como segue: de início todas as instâncias de treinamento são consideradas protótipos. Então dois conjuntos A e B, são criados. O conjunto A inicia como um conjunto vazio, enquanto B é uma cópia do conjunto de treinamento. Uma instância p, aleatória, é escolhida em B. Outra instância q, que seja a mais próxima de p é também escolhida. Caso p e q sejam da mesma classe, calcula-se um centroide[1] entre os mesmos, e o centroide é adicionado a A. As instâncias p e q são excluídas de B.

Caso ambas as instâncias sejam de classes diferentes, ambas são adicionadas a A, e excluídas de B. O processo se repete até que B seja um conjunto vazio. Neste ponto, A é copiado em B, tornando-se novamente um conjunto vazio. O algoritmo se repete até que não seja mais possível criar protótipos, ou seja, quando não é mais possível reduzir a quantidade de instâncias de B.

O VDBC é bastante semelhante ao algoritmo W*, e pode ser considerado uma variação do mesmo. Seu pseudo-código é mostrado no Algoritmo 1. As diferenças principais entre VDBC e W* são:

- A instância selecionada do conjunto de treinamento verifica entre seus vizinhos não somente outras instâncias, mas também centroides que já foram criados;
- As instâncias são verificadas sequencialmente[2], com reposição. Em outras palavras, as instâncias não são escolhidas de forma aleatória e após serem verificadas não são retiradas do conjunto de treinamento;
- O conjunto de protótipos não é reduzido ao seu máximo, ou seja, o VDBC não cria protótipos a partir de outros protótipos. A razão para isso é manter o máximo possível de protótipos, havendo redução de instâncias apenas nas regiões menos complexas do espaço amostral, para considerar as situações de desbalanceamento dos dados.

As figuras 17 e 18 apresentam exemplos do funcionamento do VDBC. A observação sobre a primeira figura deve começar na parte superior esquerda. O conjunto A, de protótipos, é iniciado vazio, enquanto o conjunto B é o conjunto de treinamento. A primeira

[1] Literalmente a média aritmética entre os pontos.
[2] Ou seja, de acordo com a sequência do *input*.

Algoritmo 1: Pseudo-código do VDBC

```
1: /*Dadas todas as instâncias de treinamento*/
2: para cada instância faça
3:    Verifique o vizinho mais próximo
4:    se vizinho mais próximo não é centroide então
5:       se vizinho mais próximo é da mesma classe então
6:          Criar centroide entre os mesmos
7:       senão
8:          Esta instância se torna um centroide
9:       fim do se
10:   senão
11:      se vizinho mais próximo é de uma classe diferente então
12:         Esta instância se torna um centroide;
13:      fim do se
14:   fim do se
15: fim do para
16: /*Dadas todas as instâncias de teste*/
17: para cada instância faça
18:    Classifique de acordo com o centroide mais próximo
19: fim do para
```

instância é selecionada e seu vizinho mais próximo é encontrado. Neste caso, ambas as instâncias fazem parte da mesma classe e, portanto, seu centroide é adicionado em A como protótipo. A instância seguinte é verificada e, como a mesma faz parte de uma classe diferente da do seu vizinho mais próximo, ambas são adicionadas como protótipos em A. A terceira instância é selecionada, entretanto seu vizinho mais próximo é também um centroide. Neste caso, se ambos forem da mesma classe, nada é feito, devido à já existência do protótipo. Contudo, se forem de classes diferentes, a instância é adicionada em A como protótipo.

A Figura 18 apresenta o VDBC gerando protótipos em uma base sintética com cinco classes desbalanceadas, e um total de mil instâncias. A primeira imagem (a) apresenta a base original. A imagem seguinte (b) apresenta o início do algoritmo, enquanto somente alguns protótipos foram criados. Como a geração é feita a partir de uma verificação sequencial, de início, somente os protótipos de uma classe são criados. As imagens seguintes (c), (d) e (e) apresentam a evolução do conjunto de protótipos até ao conjunto final (f), o qual é utilizado pelo 1NN para classificação.

O VDBC foi testado em vinte e uma bases do repositório UCI (DHEERU; TANISKIDOU, 2017). As bases estão resumidas nas Tabelas 5 e 6. As mesmas bases foram usadas em todos os testes de todas as versões do VDBC, como também no algoritmo DCIA (*Dynamic Centroid Insertion and Adjustment*), evolução do VDBC, e suas modificações.

A Tabela 5 possui seis colunas. A primeira coluna contém os nomes das bases; a segunda, terceira e quarta colunas listam a quantidade de classes, atributos e instâncias

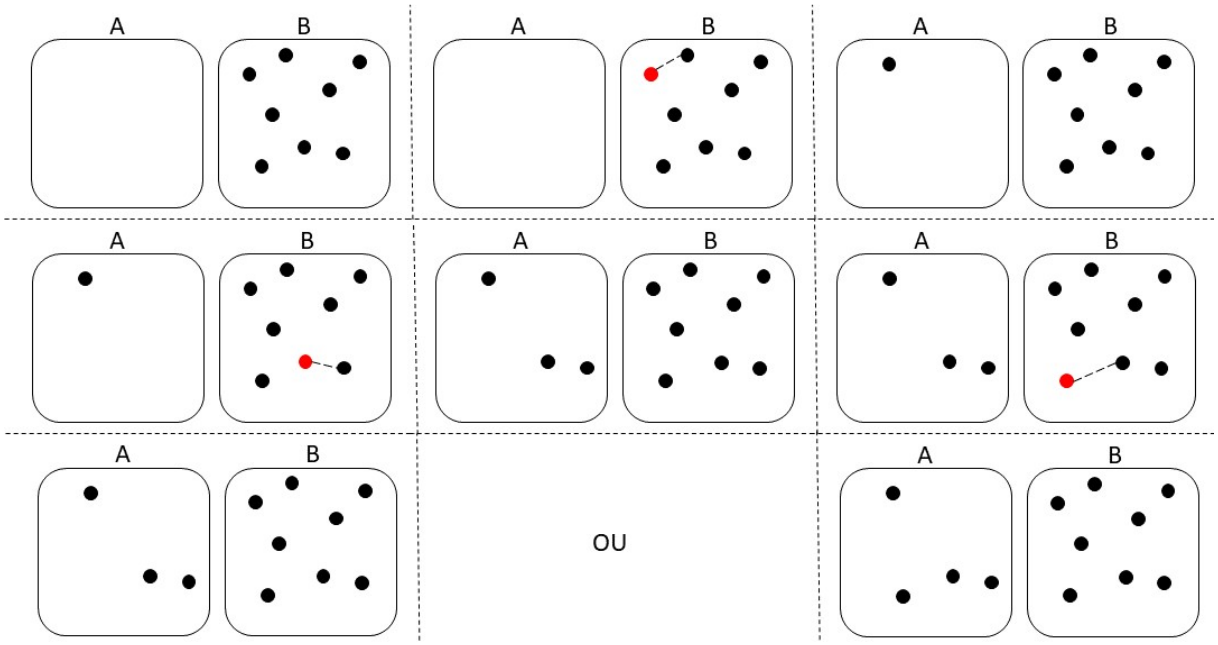

Figura 17 – Exemplo simples do funcionamento do VDBC. Começando do topo e seguindo da esquerda para a direita. Para cada instância verificada, destacada em vermelho, seu vizinho mais próximo é encontrado e então é feita a geração de protótipos.

presentes em cada base, respectivamente. A quantidade de instâncias por classe é listada na Tabela 6. As duas últimas colunas da Tabela 5 mostram os valores da taxa de desbalanceamento (IR, do Inglês *Imbalance Ratio*) e da taxa de desbalanceamento para múltiplas classes (MIR, do Inglês *Multiclass Imbalance Ratio*).

A taxa IR é calculada ao se dividir a quantidade de instâncias da maior classe pela quantidade de instâncias da menor classe. Por exemplo, a maior classe da base Gene possui 1655 instâncias, e a menor classe possui 767. O IR da base *Gene* é: $1655 \div 767 = 2,1578$. Esta taxa é apropriada para uma base de dados desbalanceada binária. Entretanto, no caso de uma base com múltiplas classes, a mesma deixa de ser apropriada por não considerar as demais classes. Por esta razão a taxa MIR foi proposta e faz parte das contribuições desta tese.

Levando-se em conta a mesma base de dados, o MIR é calculado da seguinte forma. A quantidade total de instâncias — ou o somatório da quantidade de instâncias de cada classe — é dividido pela quantidade de classes: $3190 \div 3 = 1063,3333$. Este valor representa a quantidade de instâncias que cada classe deveria possuir para que a base seja balanceada. Em seguida ele é dividido pela quantidade de instâncias de cada classe, ou seja, $1063,3333 \div 1655 = 0,6425$, $1063,3333 \div 768 = 1,3845$ e $1063,3333 \div 767 = 1,3864$. Desta forma é possível perceber o quão desbalanceada está cada classe em relação ao primeiro valor, possuindo mais (acima de 1) ou menos (abaixo de 1) do que deveria ter em um cenário ideal, ou seja, um cenário de base balanceada. Os novos valores são todos somados: $0,6425 + 1,3845 + 1,3864 = 3,4134$. O valor final é então subtraído pela quanti-

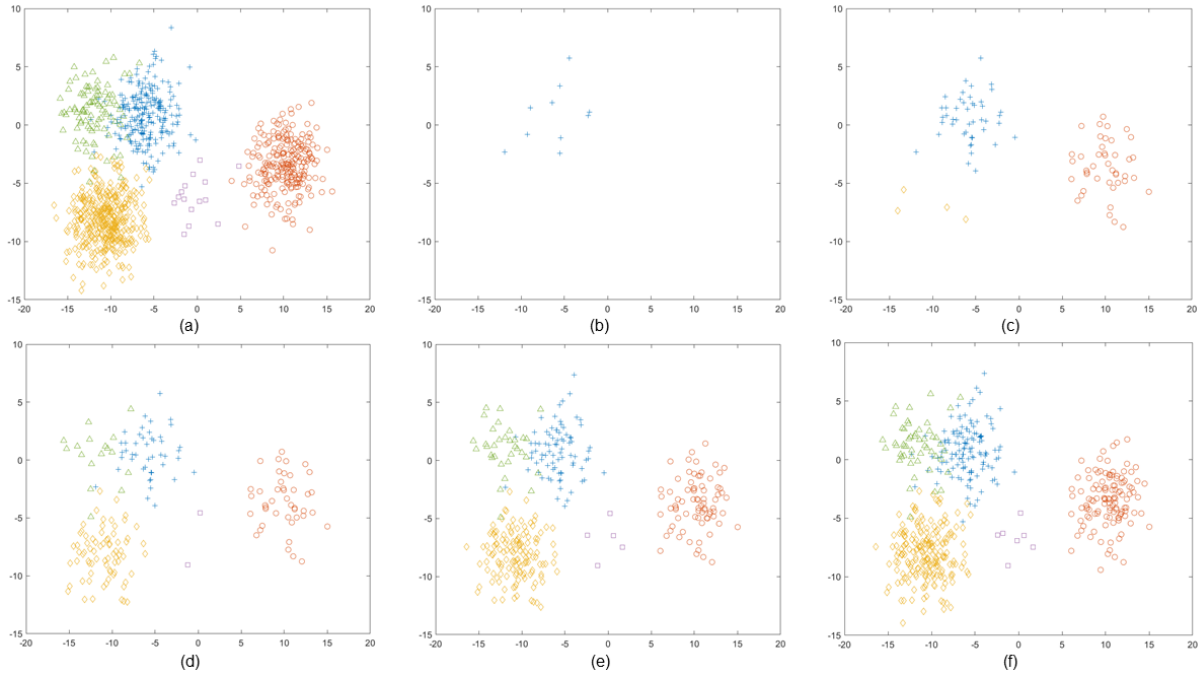

Figura 18 – Exemplo do funcionamento do VDBC sobre uma base sintética. A primeira imagem (a) apresenta a base originalmente. As imagens subsequentes apresentam a evolução do conjunto de protótipos gerados.

dade de classes: $3,4134 - 3 = 0,4134$. O valor final apresenta o grau de desbalanceamento independente da quantidade de classes.

Em alguns casos o valor da taxa MIR parece ser menor do que deveria ser. Isto acontece pelo fato de os tamanhos das classes serem relativamente homogêneos. Por exemplo, a base *Satimage* pode ser separada em dois grupos. O primeiro grupo consiste de classes com uma quantidade entre 1300 a 1500 instâncias. O segundo consiste de classes com uma quantidade entre 600 e 700 instâncias. Isto fez com que o valor de MIR fosse baixo para essa base. Ou seja, através do MIR é possível perceber se tratar de uma base razoavelmente homogênea, apesar de ser desbalanceada.

Os resultados obtidos com o VDBC são apresentados na Tabela 7. Nesta tabela os resultados são a média e desvio-padrão da métrica MAUC (HAND; TILL, 2001), média e desvio-padrão da quantidade final de protótipos e a taxa de redução, i.e., a subtração de 1 pela quantidade média de protótipos gerados dividida pela quantidade de instâncias de treinamento, ou seja 4/5 da quantidade total de instâncias. Em outras palavras, a taxa de redução apresenta a porcentagem de instâncias a menos no conjunto de treinamento.

É interessante notar que o VDBC consegue produzir[3] resultados de classificação razoáveis, ou seja, com uma média geral em torno de 0,8. Tais resultados foram alcançados utilizando, em média, pouco mais da metade da quantidade de instâncias. De forma mais

[3] Considera-se que o desempenho obtido pelo 1NN foi produzido, ou influenciado diretamente pelo VDBC.

Tabela 5 – Resumo das bases de dados usadas nos testes do VDBC e DCIA.

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Abalone	23	8	4172	344,5	268,5644
Arrhythmia	13	260	452	122,5	47,3929
Balance Scale	3	4	625	5,8776	2,6985
Car Evaluation	4	6	1728	18,6154	10,3890
Contraceptive	3	9	1473	1,8889	0,2159
Dermatology	6	34	366	5,6	1,8598
E.coli	8	7	336	71,5	47,3469
Gene	3	60	3190	2,1578	0,4134
Glass	6	9	214	8,4444	5,0133
Hayes-Roth	3	4	160	2,0968	0,3743
Horse	3	14	366	4,3269	1,2592
Nursery	5	8	12960	2160	1300,8
Page Blocks	5	10	5473	175,4643	59,5996
Post Operative	3	8	90	32	13,7188
Satimage	6	36	6435	2,4489	0,9564
Shuttle	7	9	58000	4558,6	1676,8
Soybean	15	35	630	4,6	7,6660
Thyroid	3	21	7200	40,1566	18,3396
Wine	3	13	178	1,4792	0,0774
Yeast	10	8	1484	92,6	44,7543
Zoo	7	16	101	10,25	4,9225

precisa, a quantidade de protótipos gerada, em média, correspondeu a cerca de 58% da quantidade de instâncias de treinamento das bases.

Outra informação interessante que pôde ser extraída deste experimento é que há uma forte correlação linear entre IR/MIR e a quantidade de protótipos gerados (0,9230 e 0,8207, respectivamente). Em outras palavras quanto maior o valor de IR ou MIR, maior é a quantidade de protótipos gerados. Da forma como o VDBC gera protótipos, a forte correlação citada indica que quanto maior é o desbalanceamento, mais complexo é o espaço amostral. Ainda, a correlação entre a quantidade de protótipos gerados e o valor de IR é maior que a correlação entre a quantidade de protótipos gerados e o valor de MIR. Isto sugere que a diferença de tamanho entre a maior e a menor classe tem maior efeito sobre a complexidade do espaço amostral do que a diferença de tamanho entre todas as classes.

A geração de protótipos foi realizada de forma semelhante ao W*, observando para cada instância selecionada o seu vizinho mais próximo. Contudo, pode ser possível que o mesmo processo, observando-se dois ou mais vizinhos mais próximos, gere um resultado diferente. A análise da influência da quantidade de vizinhos observados na geração de

Tabela 6 – Quantidade de Instâncias por Classe em cada Base de Dados (ordem decrescente).

Base	**#Instâncias/Classe**
Abalone	689 | 634 | 568 | 487 | 391 | 267 | 259 | 203 | 126 | 115 | 103 | 67 | 58 | 57 42 | 32 | 26 | 15 | 14 | 9 | 6 | 2 | 2
Arrhythmia	245 | 50 | 44 | 25 | 22 | 15 | 15 | 13 | 9 | 5 | 4 | 3 | 2
Balance Scale	288 | 288 | 49
Car Evaluation	1210 | 384 | 69 | 65
Contraceptive	629 | 511 | 333
Dermatology	112 | 72 | 61 | 52 | 49 | 20
E.coli	143 | 77 | 52 | 35 | 20 | 5 | 2 | 2
Gene	1655 | 768 | 767
Glass	76 | 70 | 29 | 17 | 13 | 9
Hayes-Roth	65 | 64 | 31
Horse	225 | 89 | 52
Nursery	4320 | 4266 | 4044 | 328 | 2
Page Blocks	4913 | 329 | 115 | 88 | 28
Post Operative	64 | 24 | 2
Satimage	1533 | 1508 | 1358 | 707 | 703 | 626
Shuttle	45586 | 8903 | 3267 | 171 | 50 | 13 | 10
Soybean	92 | 91 | 91 | 88 | 44 | 44 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20
Thyroid	6666 | 368 | 166
Wine	71 | 59 | 48
Yeast	463 | 429 | 244 | 163 | 51 | 44 | 35 | 30 | 20 | 5
Zoo	41 | 20 | 13 | 10 | 8 | 5 | 4

protótipos motivou a primeira modificação do VDBC, chamada VDBC.M1[4].

Os detalhes da primeira modificação, incluindo seus resultados, podem ser consultados no Apêndice A.1. O VDBC.M1 foi construído para se verificar a geração de protótipos com k vizinhos diferentes, com $k = 1, ..., 5$. O conceito geral da criação dos protótipos foi mantido, ou seja, se o(s) vizinho(s) da instância selecionada for(em) da mesma classe, um protótipo é gerado; caso contrário, a instância corrente é transformada em protótipo. Entretanto, o algoritmo agora passa a verificar três casos distintos: (1) quando todos os k vizinhos são protótipos, (2) os k vizinhos são protótipos e outras instâncias, e (3) todos os vizinhos são instâncias.

O primeiro caso a ser observado é uma novidade dessa modificação. Os dois casos seguintes já são tratados no VDBC. Com esta novidade o VDBC passa a gerar protótipos a partir de outros protótipos. Ou seja, se todos os protótipos forem da mesma classe,

[4] É importante ressaltar que houve modificação no algoritmo, e não somente alteração no valor de um parâmetro.

Tabela 7 – Desempenho do VDBC através das métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8178±0,0169	2911±7,7782	0,1278
Arrhythmia	0,7765±0,0295	247,8±3,4205	0,3147
Balance Scale	0,7180±0,0461	253,6±2,0736	0,4928
Car Evaluation	0,8763±0,0438	540,6±8,4439	0,6089
Contraceptive	0,5733±0,0168	795,4±6,2290	0,3250
Dermatology	0,9204±0,0221	156,2±3,7014	0,4665
E.coli	0,8783±0,0312	181,4±5,4589	0,3251
Gene	0,6972±0,0163	1400,6±18,3112	0,4512
Glass	0,8677±0,0310	115,4±2,8810	0,3259
Hayes-Roth	0,7305±0,0829	53,8±2,2804	0,5797
Horse	0,6644±0,0196	199,8±1,3038	0,3176
Nursery	0,8333±0,0292	3101,4±29,7035	0,7009
Page Blocks	0,7498±0,0221	2693,8±14,5842	0,3848
Post Operative	0,6291±0,0980	47,2±2,0494	0,3444
Satimage	0,9470±0,0058	2625±11,4018	0,4901
Shuttle	0,8794±0,0286	26011±36,7655	0,4394
Soybean	0,9676±0,0146	270,2±5,3572	0,4460
Thyroid	0,7740±0,0185	3609,6±9,1542	0,3733
Wine	0,8035±0,0675	94,8±1,3038	0,3343
Yeast	0,6792±0,0179	873,4±11,6748	0,2643
Zoo	0,9738±0,0464	32,8±2,3875	0,5941

um centroide é criado entre os mesmos e substitui os já existentes. No caso de $k = 1$, pode acontecer de o protótipo existente ser reposicionado no espaço para mais perto da instância corrente.

Com a análise dos resultados, as quais estão detalhadas no Apêndice A.1, foi possível perceber que a variação que observa os cinco vizinhos mais próximos produz os melhores resultados, ao mesmo tempo em que houve a geração de uma maior quantidade de protótipos. Uma rápida comparação entre os resultados do VDBC e VDBC.M1 com 5NN mostra que observar mais vizinhos produz resultados similares ou piores aos que já haviam sido obtidos com o VDBC. Portanto, para que haja uma melhora de desempenho com uma geração mais eficiente de protótipos é necessária outra estratégia.

Ao se pensar em outra estratégia para ser utilizada com o VDBC, foram implementadas e testadas mais quatro modificações:

- **VDBC.M2:** é baseada na ideia do SMOTE (CHAWLA et al., 2002), uma técnica de sobreamostragem para a classe minoritária. Entretanto, no contexto de múltiplas

classes, o conceito de classe minoritária se torna mais difícil, ou abrangente. Desta forma, foi desenvolvida uma estratégia para definir qual ou quais classes são consideradas as minoritárias. Uma vez definidas tais classes, as mesmas passam por um processo de sobreamostragem. Após a criação das novas instâncias, o VDBC é executado normalmente. Seus detalhes podem ser consultados no Apêndice A.2;

- **VDBC.M3:** consiste no tratamento de sobreposição de bordas. Com a Taxa Generalizada de Fisher (F_{gen}) (MOLLINEDA; SÁNCHEZ; SOTOCA, 2005; MALDONADO; MONTECINOS, 2014), é possível calcular a proximidade entre as classes. A partir disso os pares de classes próximas entre si, ou seja, cujo valor do F_{gen} seja menor ou igual a 0,15 (valor definido empiricamente), são selecionados. Para cada par selecionado novas instâncias são criadas entre as classes, e então o VDBC é executado normalmente. Seus detalhes são apresentados no Apêndice A.3;

- **VDBC.M4:** é uma alternativa à utilização do F_{gen} para o tratamento de sobreposição de bordas. Esta modificação utiliza Tomek Links (TOMEK, 1976) para detecção de bordas. Entretanto, em vez de utilizar sobreamostragem nas menores classes, nesta modificação foi utilizada a subamostragem nas maiores classes. Há ainda outra alteração importante. Até então o VDBC verifica sequencialmente cada instância de treinamento com reposição. A partir do VDBC.M4 as instâncias são selecionadas aleatoriamente, sem reposição. Os detalhes desta modificação podem ser consultados no Apêndice A.4;

- **VDBC.M5:** é a maior alteração ao algoritmo original. Nesta modificação todas as instâncias são transformadas em centroides de hiperesferas, cujos raios crescem a cada iteração. À medida que os raios das hiperesferas vão crescendo, as mesmas são forçadas a se juntar a outras, caso sejam da mesma classe, ou pararem de crescer, se forem de classes diferentes. Para que as menores classes não sejam prejudicadas pelas maiores, a seleção aleatória de instâncias é feita diretamente nas classes, as quais estão ordenadas, da menor para a maior. Ou seja, as instâncias da menor classe são as primeiras a serem selecionadas, aleatoriamente e sem reposição, para o crescimento das hiperesferas. As instâncias da maior classe, por conseguinte, são as últimas a passar pelo processo. Os detalhes desta última modificação estão presentes no Apêndice A.5.

3.2.1 Comparações entre VDBC e suas modificações

Nesta seção são apresentadas todas as comparações estatísticas efetuadas entre o VDBC e suas modificações. As comparações foram feitas por base de dados, ou seja, a cada base de dados verificou-se se os algoritmos podem ser considerados estatisticamente equivalentes quanto aos seus desempenhos.

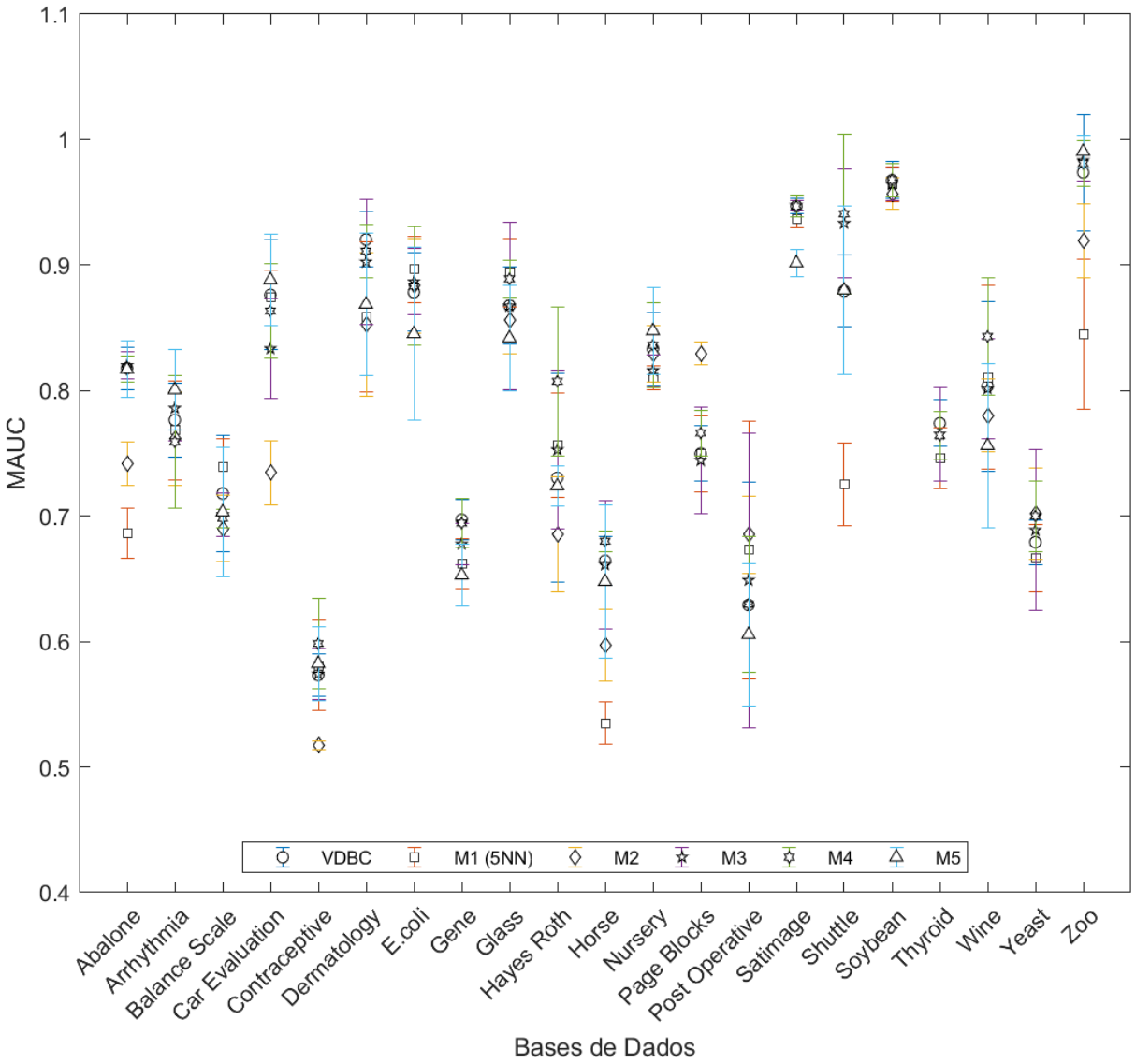

Figura 19 – Comparações visuais entre médias e desvios-padrões da métrica MAUC obtidas pelo VDBC e suas modificações.

Das variações do VDBC.M1, a que observa cinco vizinhos mais próximos foi escolhida para ser comparada às modificações do VDBC. A Figura 19 apresenta as comparações visuais por base entre o VDBC e suas modificações. O eixo das abscissas corresponde às bases de dados, enquanto o eixo das ordenadas corresponde a valores com a métrica MAUC. Esta figura é um gráfico de barras em que cada marcação significa a média obtida por algum algoritmo. As barras acima e abaixo das marcações correspondem aos desvios-padrões.

Os testes de normalidade foram feitos com as seguintes hipóteses: H_0 : *distribuição normal* e H_1 : *distribuição não normal*. As hipóteses para comparações entre três ou mais algoritmos foram: H_0 : *médias estatisticamente equivalentes* e H_1 : *médias estatisticamente não equivalentes*.

A Tabela 8 sumariza as comparações estatísticas em todas as bases. A coluna *Hipótese*

Tabela 8 – Resultados das comparações estatísticas entre VDBC e suas modificações.

Base	**Hipótese Aceita**	**p-value**
Abalone	VDBC ≡ M3 ≡ M4 ≡ M5	0,9893
Arrhythmia	Todos os desempenhos estatisticamente equivalentes	0,5533
Balance Scale	Todos os desempenhos estatisticamente equivalentes	0,2296
Car Evaluation	VDBC ≡ M1 ≡ M3 ≡ M4 ≡ M5	0,1924
Contraceptive	VDBC ≡ M1 ≡ M3 ≡ M4 ≡ M5	0,6583
Dermatology	Todos os desempenhos estatisticamente equivalentes	0,2197
E.coli	Todos os desempenhos estatisticamente equivalentes	0,6813
Gene	VDBC ≡ M3 ≡ M4	0,2057
Glass	Todos os desempenhos estatisticamente equivalentes	0,2986
Hayes-Roth	VDBC ≡ M1 ≡ M3 ≡ M4 ≡ M5	0,2018
Horse	VDBC ≡ M3 ≡ M4 ≡ M5	0,6759
Nursery	Todos os desempenhos estatisticamente equivalentes	0,2509
Page Blocks	VDBC.M2 é o melhor	—
Post Operative	Todos os desempenhos estatisticamente equivalentes	0,6652
Satimage	VDBC ≡ M3 ≡ M4	0,9858
Shuttle	VDBC ≡ M3 ≡ M4 ≡ M5	0,1607
Soybean	Todos os desempenhos estatisticamente equivalentes	0,7187
Thyroid	Todos os desempenhos estatisticamente equivalentes	0,4101
Wine	Todos os desempenhos estatisticamente equivalentes	0,2702
Yeast	Todos os desempenhos estatisticamente equivalentes	0,2647
Zoo	VDBC ≡ M3 ≡ M4 ≡ M5	0,7762

Aceita apresenta os resultados dos testes sobre os algoritmos.

Para algumas bases as comparações estatísticas confirmam o que pode ser percebido visualmente. Por exemplo, nas bases *Abalone* e *Car Evaluation* os algoritmos VDBC.M1 e VDBC.M2, respectivamente, possuem desempenhos piores que os demais.

Entretanto existem alguns casos relativamente complexos. Por exemplo, na base *Balance Scale* ao se levar em conta todos os seis desempenhos é possível afirmar que os mesmos são estatisticamente equivalentes. Contudo, se for feito um teste estatístico somente entre VDBC.M1 e VDBC.M4 temos que os desempenhos podem ser considerados estatisticamente diferentes.

Outro caso semelhante é a base *Dermatology*. Visualmente a tendência é considerar como equivalentes apenas os algoritmos VDBC, VDBC.M3 e VDBC.M4. Contudo, o desvio-padrão de alguns dos resultados permite inferir, através dos testes estatísticos, que são todos equivalentes. Mais uma vez há dois resultados, VDBC.M2 e VDBC.M4, que se analisados sozinhos podem ser considerados diferentes.

A solução empregada para estes casos é sempre considerar os resultados dos testes do

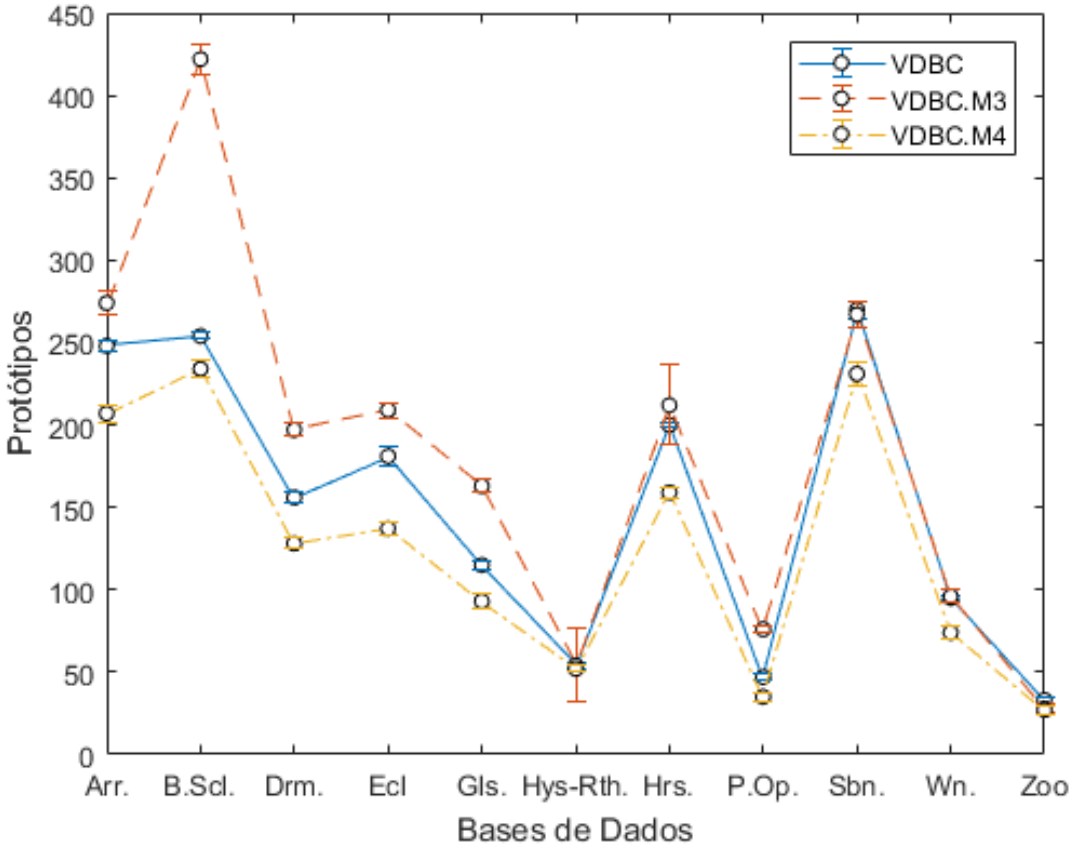

Figura 20 – Médias e desvios-padrões da quantidade de protótipos gerados pelo VDBC, VDBC.M3 e VDBC.M4 em algumas bases de dados.

ANOVA e Kruskal-Wallis, recorrendo ao *t-student* ou *ranksum* somente quando necessário. Estes foram os casos das bases *Page Blocks* e *Satimage*. Na primeira base, a evidência fornecida pelo ANOVA é que, de fato, os desempenhos não são estatisticamente equivalentes. Além disso, somente o VDBC.M2 desponta como o melhor algoritmo. Por estas razões o valor de *p-value* não foi fornecido. Quanto à segunda base, o ANOVA deixa claro que o VDBC.M5 não é estatisticamente equivalente aos demais. Entretanto o *p-value* para o teste sobre a equivalência entre VDBC, VDBC.M1, VDBC.M3 e VDBC.M4 é 0,0597, ou seja, bem próximo do limite para rejeição (i.e., $p - value < 0,05$). Entretanto, ao se realizar o teste sem o VDBC.M1, o valor do *p-value* sobe para 0,9858, sendo esta escolhida como a *Hipótese Aceita*. A razão disso são os baixos valores dos desvios-padrões, que faz com que mesmo médias muito próximas sejam praticamente consideradas como estatisticamente diferentes.

Dos algoritmos que obtiveram resultados em todas as bases de dados, o VDBC, VDBC.M3 e VDBC.M4 são estatisticamente equivalentes em 20 das 21 bases. Os três algoritmos só não alcançaram o melhor resultado em uma única base (*Page Blocks*).

Dos três algoritmos citados, o VDBC.M4 é aparentemente o mais estável possuindo, em média, o menor valor de desvio-padrão. Além disso, as comparações estatísticas sobre a

quantidade de protótipos gerados mostra que o VDBC.M4 é o mais eficiente na redução de instâncias. A Figura 20 mostra a comparação visual da quantidade de protótipos gerados entre o VDBC, VDBC.M3 e VDBC.M4 em algumas bases de dados.

O VDBC.M5 obteve resultados em 17 bases, conseguindo ser ou estar entre os melhores em 16 das mesmas. Apenas na base *E.coli* é que a quinta modificação não alcançou um desempenho estatisticamente equivalente aos demais. O destaque desta versão é o desempenho de classificação alcançado em algumas bases com uma quantidade bastante menor de protótipos gerados. Isto indica que a estratégia de geração de protótipos pode ser repensada visando uma geração mínima dos mesmos. Por fim, esta modificação precisa ter parte de sua estratégia melhorada para que possa ser executada e obter resultados em todas as bases. A estratégia atual faz com que em alguns casos o crescimento do raio da hiperesfera seja irrisório em cada iteração, o que aumenta enormemente o custo de tempo.

A partir dos experimentos conduzidos e das comparações realizadas, foi possível chegar às seguintes conclusões:

- Em bases de dados com múltiplas classes desbalanceadas é possível obter desempenhos de classificação razoáveis e até competitivos com um algoritmo bastante simples tanto na geração dos protótipos quanto no processo de classificação;
- A diferença de tamanho entre a maior e a menor classe (IR) é um bom indicativo da complexidade dos dados da base. A homogeneidade do tamanho das classes (MIR) também é um bom indicador para isso;
- O reposicionamento dos protótipos influencia na quantidade final de protótipos gerados e no desempenho de classificação do modelo utilizado;
- A complexidade dos dados de algumas bases pode requerer uma abordagem *ad hoc*. Os resultados do VDBC.M1 nas bases *Page-Blocks* e *Yeast* sugere que separar as classes em grupos similares e adaptar o algoritmo para cada grupo seja uma alternativa a ser considerada;
- A sobreamostragem, nas formas em que foi testada, não ajuda a melhorar o desempenho de classificação, nem mesmo quando o foco está sobre as bordas entre as classes. Ao mesmo tempo esta estratégia força uma geração maior de protótipos;
- A correlação moderada entre os desempenhos do VDBC e VDBC.M3 com os valores de F_{gen}, indicam que a proximidade entre as classes é um fator que pode ser considerado na construção de um algoritmo. Entretanto, como não houve diferença significativa entre esses algoritmos, deve-se pensar em outra estratégia;
- A subamostragem de instâncias de classes maiores nas regiões de borda não melhora o desempenho do algoritmo. Entretanto permite que o mesmo desempenho seja alcançado com uma menor geração de protótipos;

- O crescimento de hiperesferas é bastante eficiente em reduzir a quantidade de instâncias. A estratégia da obtenção do valor de crescimento do raio, entretanto, precisa ser melhorada;
- Priorizar o processamento nas menores classes não necessariamente resulta em melhoria de desempenho de classificação;
- É possível a obtenção de um bom desempenho de classificação com uma quantidade ainda mais reduzida de protótipos gerados.

3.3 *DYNAMIC CENTROID INSERTION AND ADJUSTMENT* - DCIA

Duas das conclusões do estudo sobre o VDBC e suas modificações chamaram bastante atenção. A primeira é sobre o posicionamento dos centroides, e sobre como esse posicionamento influencia na quantidade de protótipos gerados e no desempenho final do algoritmo. A segunda conclusão é de que é possível alcançar bons resultados com um conjunto bastante reduzido de centroides.

A partir dessas duas conclusões surgiu a hipótese: é possível que haja geração de protótipos com ajuste de suas posições, com grande redução em sua quantidade, e ainda assim o algoritmo obter um bom desempenho de classificação?

López *et al.* (LÓPEZ et al., 2014), com seu algoritmo IPADE-ID, mostraram que a resposta é sim. Entretanto, o IPADE-ID foi testado apenas sobre bases desbalanceadas binárias. É interessante, portanto, verificar se esta estratégia obtém bons resultados quando as bases possuem múltiplas classes desbalanceadas.

A ideia básica do IPADE-ID, que também foi testada no DCIA, é dividir o algoritmo em três fases. Na primeira fase, ou inicialização, a geração de protótipos começa com apenas um protótipo para cada classe. Na segunda fase há uma tentativa de ajuste de posições para uma melhoria de desempenho. A terceira fase consiste na tentativa de adição de mais protótipos, também visando uma melhoria de desempenho.

Apesar da estratégia geral ser a mesma, ambos os algoritmos foram construídos de forma diferente. Começando pela população de soluções, o DCIA considera cada conjunto de protótipos como um indivíduo diferente, enquanto o IPADE-ID considera como indivíduos os próprios protótipos. Desta forma, enquanto no primeiro algoritmo vários conjuntos de protótipos são ajustados em cada iteração, no segundo apenas um conjunto de protótipos é ajustado no espaço amostral. Uma vez que o ajuste é feito com um algoritmo de busca, a utilização de um único conjunto, como no IPADE-ID, pode apresentar uma vantagem e uma desvantagem ao mesmo tempo. A vantagem está sobre o custo de processamento, enquanto a desvantagem é um possível maior gasto de tempo no processo de busca. Em outras palavras, uma demora para a convergência em uma boa região de solução. Pode ser citada ainda outra desvantagem do IPADE-ID. Como é utilizado ape-

Algoritmo 2: Pseudo-código do DCIA

Inputs: Conjunto de treinamento, S como conjunto de soluções , N como quantidade de soluções, C como quantidade de classes

Inicialização
para $j = 1$ *a* N **faça**
  se $j == 1$ **então**
    Para cada classe, encontre o seu respectivo centroide;
  senão
    Selecione uma instância aleatória da classe como protótipo;
  fim do se
fim do para

Ajuste e Adição de Protótipos
para $i = 1$ *a Critério de Parada* **faça**
  Execute o algoritmo de busca para ajustar os protótipos de cada solução S_n;

  Crie S' como uma cópia do conjunto de soluções S;
  Selecione a melhor solução de S';
  para $c = 1$ *a* C **faça**
    Selecione uma instância aleatória e adicione à solução S'_{melhor};
    Verifique o desempenho de S'_{melhor} com o conjunto de treinamento;
    se *desempenho com adição* $>$ *desempenho sem adição* **então**
      Selecione uma instância aleatória para todas as outras soluções;
      Sobrescreva S com S';
    fim do se
  fim do para
fim do para

nas um único conjunto de protótipos, este algoritmo pode estar sujeito a ficar preso mais facilmente em ótimos locais.

A inicialização dos algoritmos pode ser diferente, dependendo do classificador utilizado. Caso seja o classificador 1NN, ambos os algoritmos possuem uma inicialização idêntica, i.e., o conjunto de protótipos é iniciado com os centroides de cada classe. Contudo, o IPADE-ID também utiliza o classificador C4.5 (QUINLAN, 1993), forçando o algoritmo a possuir uma inicialização *ad hoc*. Há ainda um detalhe sobre a inicialização do DCIA. Como são vários conjuntos de protótipos, apenas o primeiro conjunto é iniciado com os centroides de cada classe. Os demais conjuntos são iniciados ao se escolher, de forma aleatória, alguma instância de cada classe como seu primeiro protótipo.

A seleção de um bom conjunto inicial de protótipos é importante, uma vez que tal conjunto pode guiar a busca a soluções promissoras (LÓPEZ et al., 2014; TRIGUERO; GARCÍA; HERRERA, 2010). Os centroides das classes podem ser considerados bons representantes, já que são a média de todos os valores de uma determinada classe. Ainda, além de uma boa seleção inicial de protótipos, pode ser uma boa prática permitir uma busca mais am-

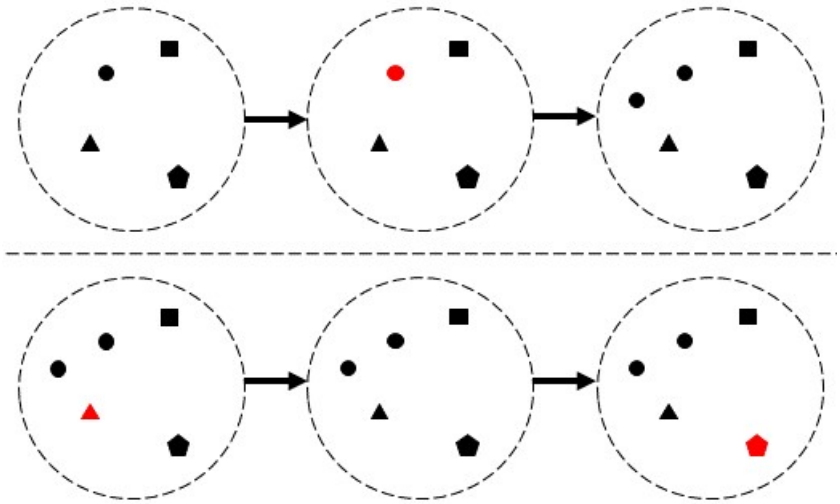

Figura 21 – Exemplo de inserção de novos protótipos no DCIA.

pla desde o começo do processo iterativo. Levando em conta que no DCIA cada conjunto de protótipos é um indivíduo, a utilização dos demais indivíduos para uma busca mais ampla pode se mostrar um fator importante.

Para a segunda fase, i.e., o ajuste dos protótipos, foram utilizados junto ao DCIA quatro algoritmos de busca: GSA (*Gravity Search Algorithm*) (RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009), SGSA (*Simple Gravity Search Algorithm*), PSO (*Particle Swarm Optimization*) (KENNEDY; EBERHART, 1995) e CSO (*Competitive Swarm Optimizer*) (CHENG; JIN, 2015). Além dos quatro algoritmos citados, qualquer outro algoritmo de busca pode também ser utilizado nesta fase, inclusive a Evolução Diferencial (STORN; PRICE, 1997) utilizada no IPADE-ID. Os detalhes de cada algoritmo de busca e como foram utilizados para ajuste de posição dos protótipos são detalhados em suas respectivas seções.

A terceira fase, que consiste na adição de mais protótipos, também foi implementada de maneiras diferentes nos algoritmos IPADE-ID e DCIA. No primeiro, as classes são marcadas como *otimizáveis*, e a que estiver produzindo menor desempenho é escolhida para crescer. Uma instância aleatória é escolhida e inserida junto aos protótipos já existentes, forçando um novo ajuste de posições. Caso o novo conjunto obtenha um melhor desempenho de classificação, o mesmo substitui o conjunto antigo. Do contrário, o novo conjunto é descartado, uma variável que guarda a quantidade de iterações sem melhoria de desempenho é atualizada, e, dependendo desse valor, a classe corrente pode ser marcada como *não-otimizável*. O processo é repetido até que nenhuma das classes sejam otimizáveis ou o valor máximo de desempenho seja alcançado.

Quanto ao DCIA, a busca pela simplicidade continua. Na terceira fase, para cada classe uma instância aleatória é escolhida e inserida no conjunto de protótipos, sem que haja novo ajuste de posições. A instância permanecerá no grupo caso sua adição melhore o desempenho do algoritmo. Caso contrário, a instância é descartada. A Figura 21 apresenta

um exemplo de como a inserção acontece. O conjunto de protótipos no canto superior esquerdo apresenta um protótipo de cada classe logo após o ajuste de posições. Em seguida há uma tentativa de inserção de um novo protótipo para a classe *círculo*. A instância aleatória selecionada contribui para o aumento de desempenho e, portanto, é inserida como novo protótipo. O passo seguinte é a tentativa de inserção de um protótipo para outra classe, desta vez a classe *triângulo*. A instância aleatória selecionada não contribuiu para o aumento de desempenho e, por isso, nenhum novo protótipo foi adicionado para esta classe. A terceira fase então continua até que a tentativa de inserção seja conduzida em todas as classes.

Após a terceira fase, haverá uma repetição da segunda e terceira fases até que se atinja um critério de parada. Normalmente o critério de parada utilizado para o DCIA é a execução de 50 iterações. A quantidade de iterações foi escolhida empiricamente.

É importante lembrar também que a estratégia de adição de protótipos no DCIA é executada somente sobre o conjunto de protótipos com melhor desempenho. Sempre que uma nova instância de uma determinada classe é adicionada, todos os outros conjuntos adicionam uma instância aleatória da mesma classe. Desta forma, garante-se que todas as soluções terão a mesma quantidade de protótipos de cada classe.

O pseudo-código do DCIA é apresentado no Algoritmo 2. Nas próximas seções são apresentadas as análise do DCIA utilizando os algoritmos de busca citados.

3.3.1 *DCIA utilizando Gravity Search Algorithm*

O modelo de classificação DCIAGSA utiliza o algoritmo de busca GSA (*Gravity Search Algorithm*) (RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009) na segunda fase do algoritmo para ajustar as posições dos protótipos. O GSA é um algoritmo recente de busca em enxame flexível e bem balanceado que realça habilidades de exploração e explotação (REZAEI; NEZAMABADI-POUR, 2015). O mesmo foi inspirado nas leis Newtonianas de gravitação e movimento. Neste algoritmo as interações entre as massas são simuladas e os objetos se movem pelo espaço de busca sob influência da gravidade (REZAEI; NEZAMABADI-POUR, 2015; RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009).

Todos os objetos se atraem pela força gravitacional. Esta força causa o movimento de todos os objetos globalmente em direção aos objetos com maiores massas, os quais correspondem às boas soluções para o problema. Formalmente, considere N objetos, onde cada um é mapeado no espaço de busca como segue:

$$X_i = \{x_i^1, x_i^2, ..., x_i^d\}; \tag{3.1}$$

no qual x_i^d é a posição do *i-ésimo* ($i = 1, ..., N$) objeto na *d-ésima* dimensão. Ainda, cada objeto possui seu próprio valor de aptidão (ou *fitness*) e massa. Neste trabalho a aptidão

é calculada a partir da métrica MAUC (HAND; TILL, 2001). A massa de um objeto na iteração t é calculada da seguinte forma:

$$M_i(t) = \frac{fitness_i(t) - worst(t)}{\sum_{j=1}^{N}(fitness_j(t) - worst(t))} \tag{3.2}$$

onde $worst(t)$ é o pior valor de aptidão obtido pelos objetos na iteração t.

Cada objeto será influenciado por uma força gravitacional (Equação 3.3). Esta força é a soma da interação gravitacional do objeto corrente e todos os outros objetos. A partir dessa força haverá uma aceleração resultante (Equação 3.4), a qual é utilizada para calcular a velocidade do objeto corrente (Equação 3.5). Por fim, a nova posição do objeto é calculada com a soma de sua posição atual e sua velocidade (Equação 3.6).

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j^d G(t) \frac{M_j(t) \times M_i(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)) \tag{3.3}$$

$$a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} = \sum_{j \in Kbest, j \neq i} rand_j^d G(t) \frac{M_j(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)) \tag{3.4}$$

$$v_i^d(t+1) = rand_i^d \times v_i^d(t) + a_i^d(t) \tag{3.5}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1) \tag{3.6}$$

Nas equações 3.3 a 3.6 $rand$ é um número aleatório gerado por uma função de distribuição uniforme no intervalo $[0, 1]$, $G(t)$ é a constante gravitacional na iteração t, ε é um valor pequeno e R_{ij} é a distância Euclidiana entre dois objetos i e j. $Kbest$ é o conjunto dos primeiros K objetos com os melhores valores de aptidão e maiores massas. A constante gravitacional é calculada da seguinte forma:

$$G = G_0(1 - \frac{t}{T}) \tag{3.7}$$

na qual o valor de G_0 é 1, t é a iteração corrente e T é a quantidade total de iterações.

O fucionamento do GSA é explicado visualmente na Figura 22. Nela existem onze objetos, no qual o primeiro, referenciado como M_1 e destacado em vermelho, é o que possui a maior massa. As setas partindo de cada objeto representam o quanto cada um é influenciado pelos demais. Observe que as maiores setas estão em direção ao objeto M_1, ou seja, o que possui a maior massa.

Neste trabalho duas alterações foram realizadas na Equação 3.3. A primeira está no denominador da divisão do produto das massas. Os autores do GSA não sugerem qualquer valor para a variável ε (RASHEDI; NEZAMABADI-POUR; SARYAZDI, 2009), portanto, no denominador foi utilizado a equação original: $(R_{ij}(t))^2$. A segunda alteração foi realizada

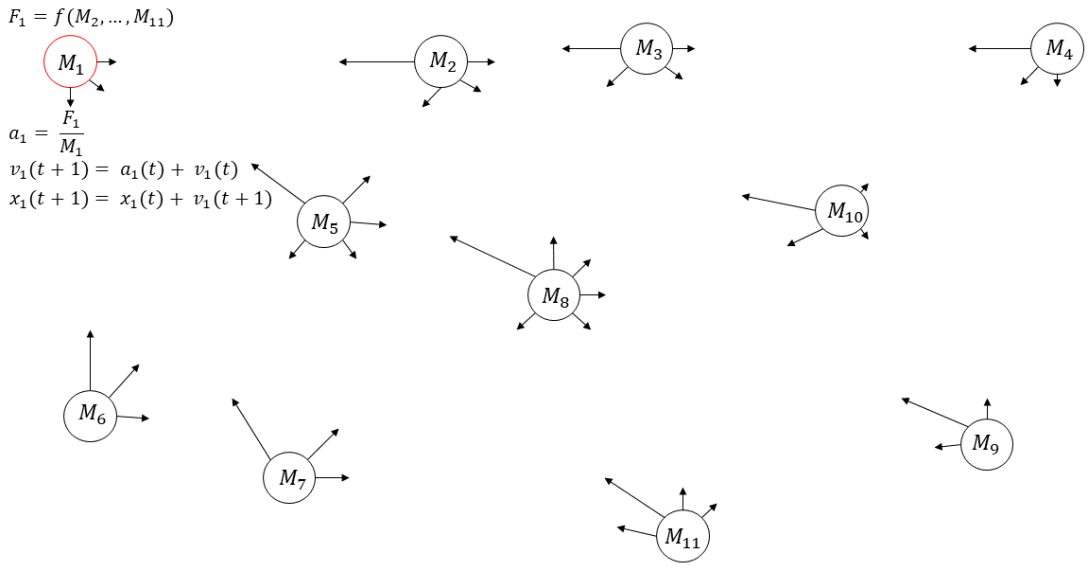

Figura 22 – Exemplo do funcionamento do GSA.

no $Kbest$. Esta variável diminui linearmente até alcançar o valor 1, o que significa que durante a execução chegará o momento em que apenas um objeto estará influenciando todos os demais. Este fator pode ser interessante para fins de convergência, contudo, pode acabar se tornando um mecanismo que deixe o algoritmo preso em um ótimo local. A diminuição linear foi excluída e o valor de $Kbest$ foi mantido como N, ou seja, durante toda a execução todos os objetos influenciam uns aos outros.

O pseudo-algoritmo do GSA é apresentado no Algoritmo 3. O critério de parada é a variável T da Equação 3.7. No DCIAGSA a instrução da linha 13 no Algoritmo 2 se confunde com a instrução da linha 2 no Algoritmo 3. Em outras palavras a segunda e terceira fases do DCIA são executadas T vezes. O valor de T foi configurado empiricamente para 50.

A Figura 23 apresenta um exemplo do funcionamento do DCIAGSA. A primeira imagem (a) apresenta a base original. A imagem seguinte (b) apresenta o DCIA no seu início, ou seja, com o centroide de cada classe como seu protótipo inicial. As imagens (c), (d) e (e) apresentam a continuação do algoritmo, com o reposicionamento e acréscimo de novos protótipos. A última imagem (f) apresenta um dos conjuntos finais de protótipos. O conjunto final que produzir o melhor desempenho com as instâncias de treinamento é o escolhido para ser usado pelo 1NN durante o teste.

Os testes foram feitos de acordo com o descrito no início deste capítulo. As bases usadas nos testes do VDBC são as mesmas usadas para o DCIAGSA, e estão resumidas nas Tabelas 5 e 6. Os resultados obtidos com a execução do DCIAGSA são apresentados na Tabela 9. Os resultados do DCIAGSA foram comparados aos do VDBC.M4, pois além

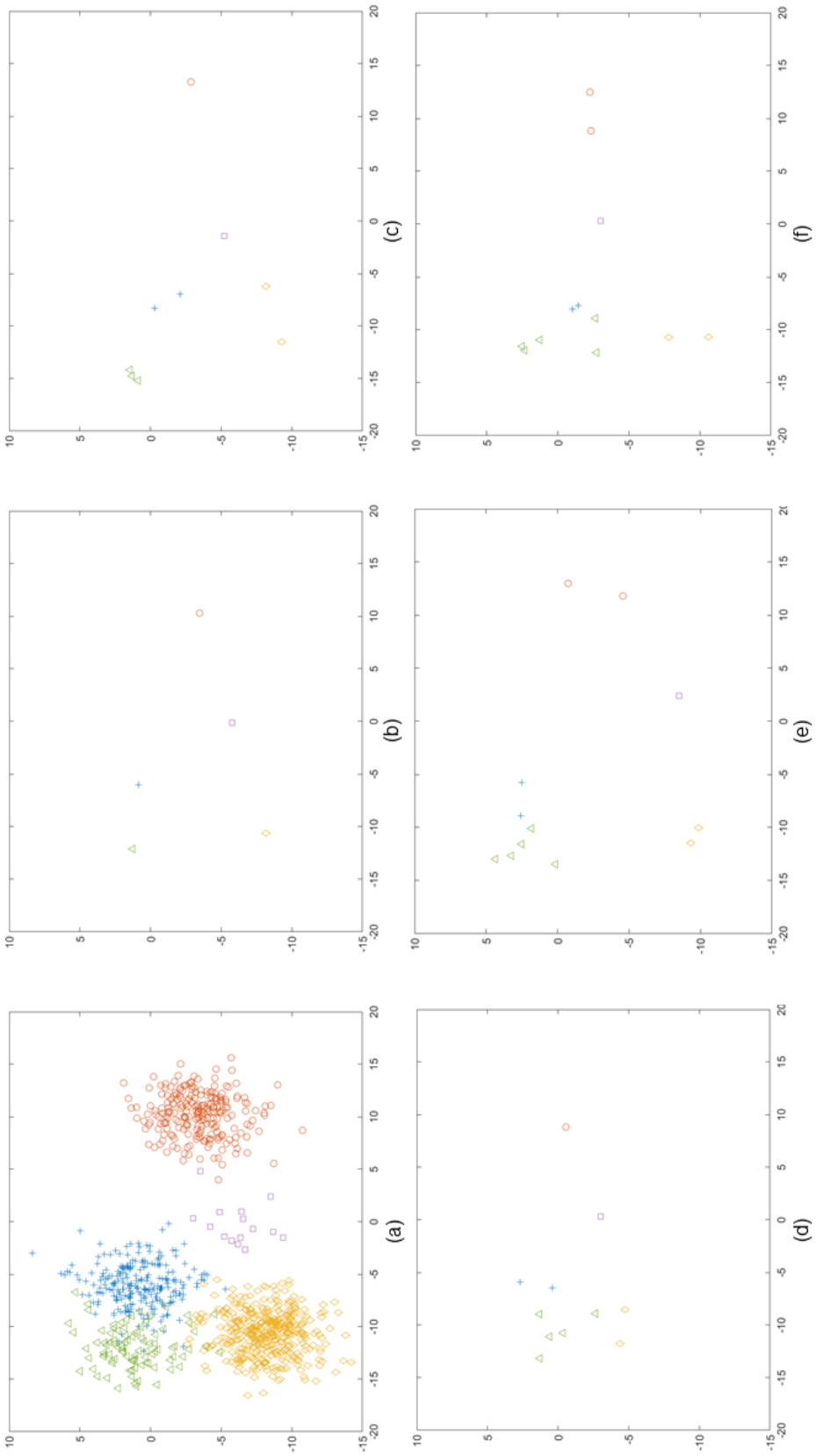

Figura 23 – Exemplo do funcionamento do DCIAGSA sobre uma base sintética. A primeira imagem (a) apresenta a base original. As imagens subsequentes apresentam a evolução do conjunto de protótipos gerados.

Algoritmo 3: Pseudo-código do GSA

```
1: Inputs: N como o tamanho da população
2: enquanto não atingir critério de parada faça
3:    para i = 1 a N faça
4:       Avalie a solução com a função de aptidão;
5:       Calcule Massa, Força, Aceleração e Velocidade usando as Equações 3.3 a 3.6;
6:    fim do para
7: fim do enquanto
```

Tabela 9 – Desempenho do DCIAGSA nas métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8201±0,0086	337±42,4382	0,8990
Arrhythmia	0,7950±0,0479	74,4±6,8775	0,7942
Balance Scale	0,7610±0,0495	8,2±4,2071	0,9836
Car Evaluation	0,9205±0,0119	29±3	0,9790
Contraceptive	0,6010±0,0335	24±9,9247	0,9796
Dermatology	0,8759±0,0252	72,6±19,3598	0,7520
E.coli	0,9074±0,0251	25,8±3,7014	0,9040
Gene	0,7432±0,0171	46,6±1,9494	0,9817
Glass	0,8877±0,0558	32,6±8,3546	0,8096
Hayes-Roth	0,7429±0,1153	41,4±3,9115	0,6766
Horse	0,6787±0,0408	15,8±6,7231	0,9460
Nursery	0,7724±0,0637	50,8±9,2304	0,9951
Page Blocks	0,7884±0,0212	21,4±4,7749	0,9951
Post Operative	0,6586±0,0608	36,6±3,6469	0,4917
Satimage	0,9272±0,0028	84,2±9,1761	0,9836
Shuttle	0,9941±0,0022	113,4±21,3026	0,9976
Soybean	0,9687±0,0096	106,6±33,3362	0,7885
Thyroid	0,7899±0,0323	17,6±8,4439	0,9969
Wine	0,8101±0,0438	5,4±1,6733	0,9621
Yeast	0,8455±0,0285	117,8±9,0940	0,9008
Zoo	0,9905±0,0130	22±5,3852	0,7277

de ter obtido um dos melhores desempenhos, possui resultados sobre todas as bases.

É interessante notar que a quantidade de protótipos gerados foi bastante reduzida. A média da *Redução*, i.e., a porcentagem da quantidade a menos de instâncias do conjunto de treinamento, para o DCIAGSA ficou em 0,1169. Ou seja, em média, os conjuntos de treinamento foram reduzidos a quase 10% do seu tamanho original. Ao mesmo tempo, as comparações com o VDBC.M4 mostram que em treze bases de dados o desempenho permaneceu estatisticamente equivalente, porém houve melhoria em quatro bases (*Balance*

Scale, *Car Evaluation*, *Gene* e *Yeast*). Apenas em duas bases (*Dermatology* e *Satimage*) houve redução de desempenho.

A nova estratégia possibilitou uma classificação com desempenho semelhante ao do VDBC.M4, porém, com mais eficiência, i.e., com um conjunto menor de protótipos gerados. Ainda, houve casos em que a nova estratégia não somente foi mais eficiente como também possibilitou melhoria no desempenho.

Quanto às duas últimas bases citadas, à primeira vista, as mesmas possuem duas características em comum: a mesma quantidade de classes e uma quantidade semelhante de atributos. Entretanto há ainda outra característica comum a ambas as bases e que pode ter influenciado no desempenho: a variação dos valores dos atributos. Na base *Satimage*, por exemplo, é possível haver uma diferença absoluta nos valores de um atributo de até 128. Em outras palavras, é como se para um determinado atributo uma instância possuísse o valor 0, e outra instância, para o mesmo atributo, possuísse o valor 128. Essas diferenças podem ser prejudiciais à classificação, uma vez que atributos com larga faixa de valores podem acabar tendo um peso maior que atributos com uma faixa menor de valores (HAN; KAMBER; PEI, 2012).

Ao mesmo tempo, o valor de F_{gen} para a base *Satimage* é maior que o valor de F_{gen} para a base *Dermatology*. Ou seja, as classes da primeira base são mais separadas, o que pode ter influenciado na diminuição menor de desempenho. A queda de desempenho, em relação ao VDBC.M4, na base *Satimage* foi de cerca de 2 pontos percentuais, enquanto a queda de desempenho na base *Dermatology* foi de aproximadamente 4 pontos percentuais.

Em relação à quantidade de protótipos gerados, as comparações estatísticas mostram que o DCIAGSA gerou menos protótipos que o VDBC.M4 em praticamente todas as bases, exceto em *Post-Operative* e *Zoo*. Estas bases são as duas menores, com 90 e 101 instâncias, respectivamente. Portanto, uma maior redução da quantidade de instâncias de treinamento nessas bases pode ser desafiador. Contudo é possível, como pode ser visto com o VDBC.M5, o qual gerou menos protótipos para a base *Zoo*.

De forma análoga ao VDBC, com o DCIA também foram testadas várias modificações. Especificamente sobre o DCIAGSA duas modificações foram testadas, i.e., DCIAGSA.M1 e DCIAGSA.M2. A primeira modificação consiste na normalização dos dados anterior ao procedimento do algoritmo, com o *z-score* (HAN; KAMBER; PEI, 2012; JAIN; BHANDARE, 2011). A modificação seguinte faz a separação dos dados com uma técnica chamada DOB-SCV (*Distribution Optimally Balanced Stratified Cross-Validation*) (MORENO-TORRES; SÁEZ; HERRERA, 2012). O intuito do DOB-SCV é a separação dos dados em *k-folds* lidando tanto com o *prior-probability shift* quanto com o *covariate shift*, ambos tipos de *data set shift*. Os detalhes dessas duas modificações são apresentados nos Apêndices B.1 e B.2.

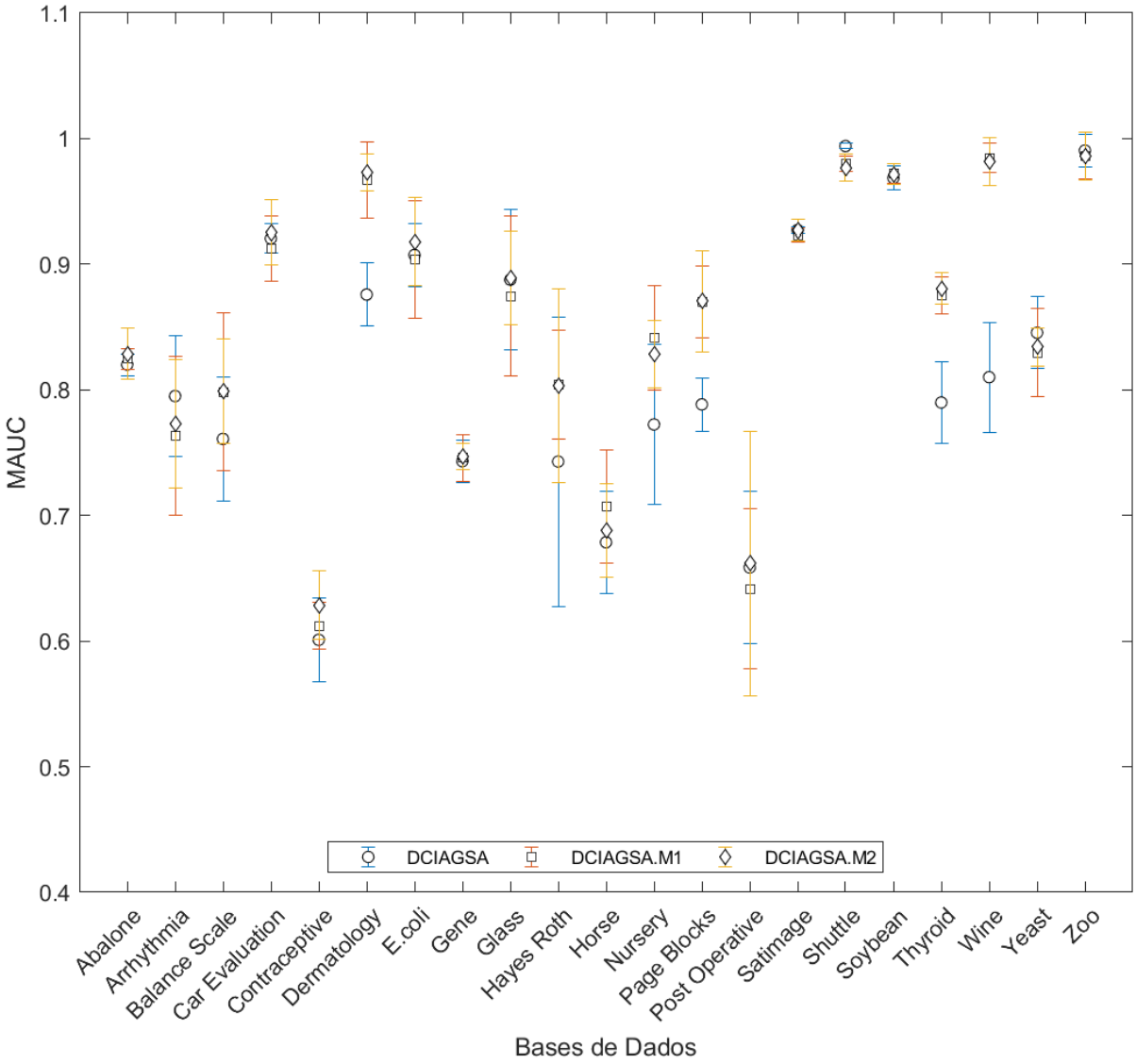

Figura 24 – Comparações visuais entre médias e desvios-padrões da métrica MAUC obtidas pelo DCIAGSA e suas modificações.

3.3.1.1 Comparações entre DCIAGSA e suas modificações

As comparações estatísticas visam verificar se o desempenho obtido com o DCIAGSA é estatisticamente equivalente aos desempenhos obtidos pelas suas duas modificações. Um teste de normalidade é executado para os desempenhos de cada algoritmo em cada base de dados. Caso todos os desempenhos façam parte de uma distribuição normal um teste paramétrico é executado. Caso contrário é executado um teste não-paramétrico.

A Figura 24 apresenta uma comparação visual entre as médias e desvios-padrões dos três algoritmos nas vinte e uma bases de dados. Cada marcação corresponde a uma média de desempenho, enquanto as barras correspondem aos desvios-padrões associados às médias.

A Tabela 10 apresenta um resumo dos testes estatísticos com a métrica MAUC. De

Tabela 10 – Resultados das comparações estatísticas entre DCIAGSA e suas modificações.

Base	**Hipótese Aceita**	**p-value**
Abalone	Todos os desempenhos estatisticamente equivalentes	0,6220
Arrhythmia	Todos os desempenhos estatisticamente equivalentes	0,6519
Balance Scale	Todos os desempenhos estatisticamente equivalentes	0,1451
Car Evaluation	Todos os desempenhos estatisticamente equivalentes	0,7788
Contraceptive	Todos os desempenhos estatisticamente equivalentes	0,3074
Dermatology	DCIAGSA.M1 ≡ DCIAGSA.M2	0,7072
E.coli	Todos os desempenhos estatisticamente equivalentes	0,8145
Gene	Todos os desempenhos estatisticamente equivalentes	0,9278
Glass	Todos os desempenhos estatisticamente equivalentes	0,8950
Hayes-Roth	Todos os desempenhos estatisticamente equivalentes	0,4409
Horse	Todos os desempenhos estatisticamente equivalentes	0,5597
Nursery	Todos os desempenhos estatisticamente equivalentes	0,0822
Page Blocks	DCIAGSA.M1 ≡ DCIAGSA.M2	0,9848
Post Operative	Todos os desempenhos estatisticamente equivalentes	0,9125
Satimage	Todos os desempenhos estatisticamente equivalentes	0,5737
Shuttle	DCIAGSA é o melhor	—
Soybean	Todos os desempenhos estatisticamente equivalentes	0,7928
Thyroid	DCIAGSA.M1 ≡ DCIAGSA.M2	0,5567
Wine	DCIAGSA.M1 ≡ DCIAGSA.M2	0,8128
Yeast	Todos os desempenhos estatisticamente equivalentes	0,6556
Zoo	Todos os desempenhos estatisticamente equivalentes	0,8550

forma análoga aos testes executados com o VDBC, a coluna *Hipótese Aceita* apresenta a conclusão dos testes, i.e., quais algoritmos são os melhores e/ou são estatisticamente equivalentes.

Na maioria das bases (i.e., em 16 das 21) não houve diferença estatística significativa entre os desempenhos dos três algoritmos. Em quatro das cinco bases restantes, a primeira modificação realizada sobre o DCIAGSA conseguiu produzir um aumento substancial de desempenho do modelo de classificação. Devido a isto, a normalização dos dados foi adotada em todos os demais algoritmos. Na base *Shuttle*, contudo, o algoritmo original teve o melhor desempenho. Entretanto, as diferenças entre os desempenhos dos três algoritmos é muito pequena, e todos possuem um desempenho superior a 97% na métrica MAUC.

A segunda modificação, i.e., dividir os dados com DOB-SCV, foi estatisticamente equivalente à primeira modificação em todas as bases. A partir desse ponto um pequeno estudo foi feito a fim de se descobrir se há outras diferenças entre o DOB-SCV e o SCV que justifique, de alguma forma, a escolha de algum dos métodos de partição.

O DOB-SCV busca diminuir o *covariate shift*, ou seja, a diferença entre as distribui-

Tabela 11 – Médias das distâncias entre centroides do conjunto de treinamento e teste na base *Balance Scale*.

Método		**Classe 1**	**Classe 2**	**Classe 3**	**Médias**
SCV	*Média*	0,8479	0,2537	0,3015	0,4677
	Desvio-Padrão	0,2699	0,1334	0,1094	0,1709
DOB-SCV	*Média*	0,4378	0,1729	0,1518	0,2541
	Desvio-Padrão	0,1292	0,0926	0,0633	0,0950

ções dos conjuntos de treinamento e teste. Então, para cada classe de cada base foram calculados centroides em ambos os conjuntos e a distância entre os mesmos. A quantidade de *folds* continuou a mesma que foi usada nos experimentos anteriores, ou seja, cinco. Portanto, para cada classe de cada base foram calculadas a média e o desvio-padrão das distâncias entre os centroides de treinamento e teste. A média das médias das classes e a média dos desvios-padrões também foram calculados para resumirem as distâncias de cada base.

Para melhor entendimento do processo, a Tabela 11 apresenta um exemplo das médias calculadas para a base *Balance Scale*. Esta base possui três classes. Para cada uma das classes foram calculados a média e o desvio-padrão da distância entre os centroides de treinamento e teste. A última coluna apresenta a média das médias e a média dos desvios-padrões. Os valores na última coluna são vistos como um *resumo* do método de particionamento na referida base.

Em posse dos *resumos* dos métodos de particionamento em todas as bases, foi possível calcular que em média o SCV deixa os centroides de treinamento das classes distantes em 1,5858 dos centroides de teste, com desvio-padrão de 0,7853. O DOB-SCV diminui esses valores para 1,3713 e 0,6912 respectivamente. Os valores conseguidos por ambos os métodos são estatisticamente equivalentes, porém, o DOB-SCV aparentemente é mais estável, ao possuir, em média, um valor menor de desvios-padrões.

Foram calculadas também as correlações lineares entre as médias obtidas por cada base e seus respectivos valores de IR, MIR e F_{gen}. Para as médias obtidas com o SCV, as correlações são, respectivamente: 0,1317, 0,0815 e -0,0739. Para o DOB-SCV os valores são, respectivamente: 0,0291, -0,0037 e -0,0595. Os valores das correlações sugerem que ambos os métodos não são influenciados pelo grau de desbalanceamento, ou por quão próximas estão as classes umas das outras.

Por fim, foram calculadas as correlações lineares entre as médias de distância de cada classe e seus respectivos tamanhos. A Tabela 12 apresenta os valores das correlações para cada base. Quanto mais próximo de 1 é o valor apresentado, maior é a correlação linear positiva, i.e., quanto mais instâncias uma classe tiver maior é a distância entre seus centroides nos conjuntos de treinamento e teste. Do contrário, quanto mais próximo

Tabela 12 – Correlações lineares entre as distâncias médias dos centroides de treinamento e teste e o tamanho das classes

Base	**SCV**	**DOB-SCV**	**Base**	**SCV**	**DOB-SCV**
Abalone	-0,5694	-0,5610	Nursery	-0,9072	-0,7485
Arrhythmia	-0,4817	-0,4870	Page Blocks	-0,8618	-0,7124
Balance Scale	-0,9974	-0,9978	Post Operative	-0,5988	-0,8143
Car Evaluation	-0,8041	-0,6620	Satimage	-0,7897	-0,8313
Contraceptive	-0,8852	-0,6185	Shuttle	-0,3394	-0,3227
Dermatology	-0,5945	-0,6094	Soybean	-0,7477	-0,7255
E.coli	-0,5608	-0,5836	Thyroid	-0,8747	-0,7699
Gene	-0,9982	-0,9995	Wine	0,0370	0,3149
Glass	-0,7274	-0,7983	Yeast	-0,5232	-0,7169
Hayes-Roth	-0,9994	-0,9992	Zoo	-0,4869	-0,5239
Horse	-0,9309	-0,9705			

o valor for de -1, maior é a correlação linear negativa. Em outras palavras, quanto mais instâncias houver em uma classe menor será a distância entre seus centroides nos conjuntos de treinamento e teste. Por fim, quanto mais próximo de 0, menor é a correlação entre os dois fatores.

Quando uma classe possui muitas instâncias é natural que, após o processo de particionamento, os conjuntos de treinamento e teste sejam bastante similares. Logo, centroides da classe em cada conjunto tendem a ficar mais próximos neste caso. Por outro lado, quanto menos instâncias existem em uma classe, fica mais difícil manter as distribuições o mais similar possível. Por isso, nesses casos, pode ser mais comum que centroides da classe nos conjuntos de treinamento e teste fiquem mais distantes. Apesar de contribuir bastante, o tamanho da classe não é o único fator que influencia na distância dos centroides. A variação dos valores também possui um papel importante. Quanto mais densa é uma classe, ou seja, quanto menor o desvio-padrão, maior é a possibilidade de os centroides estarem mais próximos.

Na Tabela 12 a maior parte das correlações é negativa. A partir disso é possível perceber a tendência dos centroides ficarem mais próximos se as classes possuem muitas instâncias. A correlação se torna positiva, normalmente, quando as instâncias das maiores classes são mais dispersas, ou seja, possuem um valor maior de desvio-padrão. Contudo, talvez seja interessante que tal correlação seja o mais próximo possível de zero, principalmente no caso de classes desbalanceadas. Desta forma os centroides de todas as classes nos conjuntos de treinamento e teste estariam mais próximos independente do tamanho da classe.

A média e desvio-padrão das correlações obtidos com SCV é $-0,6972 \pm 0,2580$. A média e desvio-padrão obtidos com DOB-SCV é $-0.6732 \pm 0,2892$. A base *Wine* foi a

única em que ambos os métodos de particionamento obtiveram uma correlação positiva, apesar de quase nula ou fraca. Nesta base, após o particionamento com o SCV a média das distância para todas as classes é bem próxima. O DOB-SCV, contudo, foi um pouco afetado pela dispersão das instâncias da maior classe, resultando em uma maior correlação positiva, ainda que fraca. Ao se considerar a correlação obtida na base *Wine* como *outlier*, as médias e desvios-padrões obtidos pelo SCV e DOB-SCV se tornam, respectivamente: -0.7339 ± 0.2007 e -0.7226 ± 0.1845. As médias são estatisticamente equivalentes com $p - value = 0,8538$.

Em conclusão, para as bases testadas, que possuem três ou mais classes desbalanceadas, o DOB-SCV não apresentou qualquer tipo de melhoria evidente em relação ao SCV. Apesar disso, o novo método aparenta ser um pouco mais estável. Portanto, as bases testadas não possuem um *covariate shift* natural, e também não sofrem desse problema após o particionamento. Contudo, essa conclusão diz respeito somente às vinte e uma bases testadas, quando divididas em cinco conjuntos.

3.3.2 *DCIA com Simple Gravity Search Algorithm*

O SGSA (*Simple Gravity Search Algorithm*) é proposto neste trabalho, ele é outra contribuição desta tese. Seu desenvolvimento foi motivado pela ideia de simplificar o GSA e ao mesmo tempo manter o desempenho. Seus detalhes são descritos a seguir.

Enquanto no GSA todos os objetos/soluções influenciam umas às outras, no SGSA somente a melhor solução influencia as demais. Por consequência, há uma redução de parâmetros e cálculos. Não há mais necessidade de se calcular força gravitacional, aceleração e velocidade. Estas variáveis foram substitutídas por uma única, chamada *pull*:

$$pull_i = best(t)/dist^2 \tag{3.8}$$

na qual $best(t)$ é a melhor solução na iteração t e $dist^2$ é o quadrado da distância Euclidiana entre a solução corrente e $best(t)$. A cada iteração todas as soluções irão se mover, explorando o espaço, exceto a que tiver a melhor aptidão, ou *fitness*. A *movimentação* das soluções vai depender do quão boa é a melhor solução, e do quão próximo ambas estão uma da outra. A Figura 25 apresenta um exemplo do funcionamento do SGSA. Novamente são onze objetos, no qual o primeiro é o que possui a melhor solução. Desta vez os demais objetos apontam apenas para o melhor, i.e., aquele que influencia todos os demais.

A melhor solução, entretanto, não fica inteiramente estática. Apesar de não realizar exploração no espaço, a mesma *explota* sua vizinhança em busca de melhorar seu desempenho. A explotação acontece da seguinte forma. A melhor solução é copiada. Na cópia, para cada dimensão do espaço, em cada centroide, é decidido aleatoriamente se pode haver um movimento. Caso positivo, é decidido aleatoriamente se o valor naquela dimensão será acrescido ou decrescido. Após todas as decisões serem tomadas e houver a movimentação,

Algoritmo 4: Pseudo-código do SGSA

```
1: Inputs: N como o tamanho da população e População Inicial
2: enquanto não atingir critério de parada faça
3:   para i = 1 a N faça
4:     se i não é a melhor solução então
5:       Calcule pull como na Equação 3.8;
6:       Avalie a solução com a função de aptidão;
7:     senão
8:       Faça uma cópia de best(t);
9:       para cada protótipo presente na cópia faça
10:        se rand() < 0.5 então
11:          para Cada dimensão da solução faça
12:            se rand() < 0.5 então
13:              Calcule distância de explotação: d = rand()/100;
14:              se rand() < 0.5 então
15:                Some d ao valor da dimensão;
16:              senão
17:                Diminua d do valor da dimensão;
18:              fim do se
19:            fim do se
20:          fim do para
21:        fim do se
22:      fim do para
23:      Avalie a cópia com a função de aptidão;
24:      se desempenho(cópia) > desempenho(best(t)) então
25:        substitua best(t) pela cópia.
26:      fim do se
27:    fim do se
28:  fim do para
29: fim do enquanto
```

caso a cópia tenha melhor desempenho, a mesma substitui a melhor solução. O valor que é somado ou diminuído nas dimensões é calculado da seguinte forma: um número aleatório entre 0 e 1 é escolhido e depois dividido por 100. Para melhor entendimento, o pseudo-código do SGSA é apresentado no Algoritmo 4.

A Tabela 13 apresenta os resultados obtidos pelo DCIASGSA. Na tabela estão presentes as médias e desvios-padrões dos valores obtidos com a métrica MAUC e a quantidade de protótipos gerados. Além disso, é apresentada também a taxa de redução.

As comparações do DCIASGSA foram feitas com os resultados obtidos pelo DCIAGSA.M2. Os resultados das comparações comprovam que o DCIASGSA é estatisticamente equivalente ao DCIAGSA em todos os desempenhos, e em todas as quantidades de protótipos, exceto na base *Yeast*, onde conseguiu o mesmo desempenho, porém com menos instâncias de treinamento. O DCIAGSA.M2 produziu, em média, 0,0756 protótipos, enquanto o DCIASGSA produziu 0,0735. O objetivo do DCIASGSA, portanto, foi

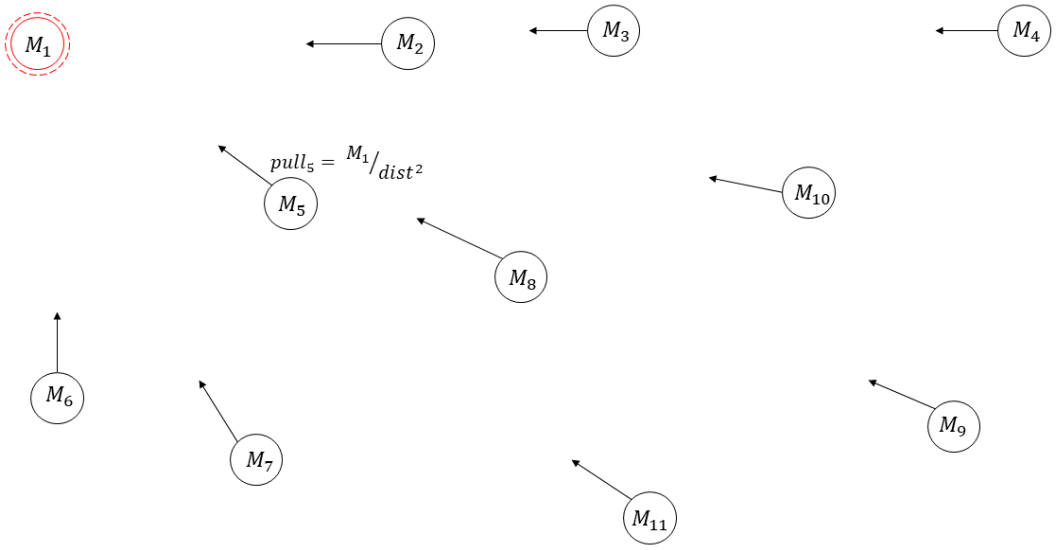

Figura 25 – Exemplo do funcionamento do SGSA.

alcançado, i.e., manter o desempenho após uma simplificação no algoritmo.

Sobre o DCIASGSA foram testadas quatro modificações. As duas primeiras consistem na utilização de técnicas de reamostragem, enquanto as duas seguintes se concentram na utilização de Seleção de Atributos.

O DCIASGSA.M1 acrescenta as mesmas técnicas utilizadas no VDBC.M4 e VDBC.M2, respectivamente, ou seja, subamostragem com Tomek Links seguido de uma sobreamostragem inspirada no SMOTE. O DCIASGSA.M2 utiliza as mesmas técnicas, entretanto em ordem inversa, ou seja, sobreamostragem e então subamostragem. Seus detalhes de implementação e resultados são apresentados no Apêndice B.3.

As duas modificações seguintes tiveram por objetivo avaliar a viabilidade da Seleção de Atributos como parte de uma solução para bases com múltiplas classes desbalanceadas. No DCIASGSA.M3 foi testado o algoritmo t-SNE (MAATEN; HINTON, 2008), enquanto no DCIASGSA.M4 foi testado o algoritmo FSCNCA, o qual é baseado no algoritmo apresentado em (YANG; WANG; ZUO, 2012). Os detalhes destas modificações e seus resultados são apresentados no Apêndice B.4.

3.3.3 *DCIA com Particle Swarm Optimization*

O DCIAPSO utililza o algoritmo de enxames PSO (KENNEDY; EBERHART, 1995) no ajuste dos protótipos gerados. O modelo de execução é o mesmo do DCIAGSA e DCIASGSA, ou seja, por 50 iterações ocorrem os passos de inserção e ajuste de protótipos. O PSO foi executado com duas pequenas modificações: (1) o parâmetro de inércia foi mantido com o valor 1, por questão de simplicidade e, (2) sempre que houve inserção de novos protótipos

Tabela 13 – Desempenho do DCIASGSA através das métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8206±0,0136	265,8±18,9130	0,9204
Arrhythmia	0,7610±0,0476	71,8±3,1145	0,8014
Balance Scale	0,8000±0,0620	7±2,8284	0,9860
Car Evaluation	0,9108±0,0129	33,8±6,0581	0,9755
Contraceptive	0,6359±0,0408	17,2±6,7602	0,9854
Dermatology	0,9721±0,0068	12,6±2,8810	0,9570
E.coli	0,9092±0,0190	28,8±3,1145	0,8929
Gene	0,7480±0,0174	47,8±6,0581	0,9813
Glass	0,8797±0,0312	31,2±3,9623	0,8178
Hayes-Roth	0,7995±0,0505	14±4,5277	0,8906
Horse	0,7027±0,0213	21±9,1104	0,9283
Nursery	0,8241±0,0358	11,2±2,8636	0,9989
Page Blocks	0,8720±0,0167	49±12,9422	0,9888
Post Operative	0,6176±0,0742	26,4±3,5071	0,6333
Satimage	0,9222±0,0065	78,8±22,0839	0,9847
Shuttle	0,9800±0,0096	53,2±9,7826	0,9989
Soybean	0,9729±0,0151	40,8±2,9496	0,9190
Thyroid	0,8792±0,0186	18,2±8,9275	0,9968
Wine	0,9848±0,0153	6±0,7071	0,9579
Yeast	0,8470±0,0286	73,8±14,3073	0,9378
Zoo	0,9587±0,0262	7,8±1,3038	0,9035

o parâmetro de velocidade foi reiniciado com o valor 0, uma vez que se trata de uma nova solução.

A Tabela 14 apresenta os resultados obtidos pelo DCIAPSO. Este algoritmo foi implementado e testado pouco após o DCIASGSA, portanto, as comparações estatísticas foram realizadas entre os dois. A partir dos resultados das comparações é possível perceber que não há diferença significativa entre os desempenhos do DCIAPSO e DCIASGSA. Entretanto utilizar o PSO para o ajuste dos protótipos possibilitou a diminuição do conjunto de treinamento em três bases: *Glass*, *Page Blocks* e *Post Operative*.

Após o primeiro teste, várias modificações foram pensadas, implementadas e testadas. O DCIAPSO foi utilizado como base de teste para as novas propostas por ter um tempo de execução menor, observado empiricamente. As modificações foram motivadas por pesquisas na literatura, ou devido aos resultados obtidos em uma modificação anterior. Ao todo foram testadas onze modificações, as quais são descritas a seguir:

- **DCIAPSO.M1:** substitui a métrica MAUC pela AUCarea durante a geração de

Tabela 14 – Desempenho do DCIAPSO através das métricas MAUC, Geração e Redução de Instâncias

Nome	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8351±0,0194	273,4±23,3088	0,9181
Arrhythmia	0,7929±0,0642	64,2±14,0961	0,8225
Balance Scale	0,7856±0,0390	8±2,3452	0,9840
Car Evaluation	0,9255±0,0302	33,4±4,4497	0,9758
Contraceptive	0,6336±0,0352	11,4±4,9295	0,9903
Dermatology	0,9706±0,0196	23±16,0312	0,9214
E.coli	0,9209±0,0243	30,2±7,5299	0,8876
Gene	0,7535±0,0213	48,8±4,6043	0,9809
Glass	0,8915±0,0297	23,8±5,7619	0,8610
Hayes-Roth	0,8074±0,0739	15,6±3,9115	0,8781
Horse	0,7037±0,0368	18±4,8990	0,9385
Nursery	0,8253±0,0474	24,8±14,5671	0,9976
Page Blocks	0,8791±0,0285	23±11,6619	0,9947
Post Operative	0,6213±0,0612	19,8±3,7014	0,7250
Satimage	0,9231±0,0061	70±6,1644	0,9864
Shuttle	0,9565±0,0210	41,6±15,9154	0,9991
Soybean	0,9654±0,0211	39,2±7,0143	0,9222
Thyroid	0,8712±0,0143	16,8±1,0954	0,9971
Wine	0,9822±0,0147	7,6±2,7019	0,9466
Yeast	0,8276±0,0241	61,4±11,0136	0,9483
Zoo	0,9700±0,0416	10,4±2,3022	0,8713

protótipos. A motivação é a maior sensibilidade da AUCarea, a qual pode pressionar o ajuste dos protótipos para uma melhor configuração. Os detalhes desta modificação estão presentes no Apêndice B.5;

- **DCIAPSO.M2:** utiliza o CSO para selecionar atributos, como sugerido em (GU; CHENG; JIN, 2018). Após a normalização dos dados, os atributos são selecionados a partir de uma abordagem *wrapper* (MOAYEDIKIA et al., 2017), a qual verifica o conjunto de atributos que produz o melhor desempenho de classificação com as instâncias de treinamento. Devido aos bons resultados, a partir de então a seleção de atributos passou a ser adotada nas modificações seguintes. Os detalhes da proposta podem ser consultados no Apêndice B.6;

- **DCIAPSO.M3:** tem por motivação verificar o desempenho do algoritmo ao se alterar a forma de calcular o *fitness* durante a seleção de atributos. Com a alteração realizada, não foram observadas diferenças estatísticas nos resultados. Detalhes sobre a alteração podem ser consultados no Apêndice B.7;

- **DCIAPSO.M4:** substitui o CSO pelo *Whale Optimization Algorithm with Crossover and Mutation operators* (WOA-CM) (MAFARJA; MIRJALILI, 2018) durante a seleção de atributos. O WOA-CM foi escolhido devido aos seus bons resultados reportados. Novamente a seleção de atributos influenciou positivamente no desempenho de classificação. Os detalhes desta modificação podem ser consultados no Apêndice B.8;

- **DCIAPSO.M5:** é similar à modificação anterior, ou seja, substitui o CSO para seleção de atributos pelo algoritmo SYMON (MOAYEDIKIA et al., 2017), o qual foi construído para selecionar atributos em bases com alta dimensionalidade. Entretanto, desta vez a seleção de atributos teve por consequência uma piora do desempenho. No Apêndice B.9 os detalhes desta modificação são apresentados;

- **DCIAPSO.M6:** altera o cáculo de *fitness* durante a seleção de atributos com o SYMON. Esta modificação pode ser vista como uma junção do DCIAPSO.M3 com o DCIAPSO.M5, e procura verificar se um cálculo de *fitness* baseado na distância entre as classes produz algum efeito positivo sobre o SYMON. O detalhamento desta modificação está presente no Apêndice B.10. Os resultados entretanto apontam que houve uma piora de desempenho, mesmo em relação ao DCIAPSO.M5. Há duas hipóteses que podem explicar o baixo desempenho produzido por estas duas últimas modificações. A primeira é sobre o uso do SYMON para seleção de atributos, de modo que, independentemente das modificações, é possível que só haja melhoria com ajustes nos parâmetros deste algoritmo. A segunda hipótese é a forma como é calculado o *fitness*;

- **DCIAPSO.M7:** verifica a primeira hipótese levantada na modificação anterior. O cálculo do *fitness* como no DCIAPSO.M6 é mantido, entretanto o SYMON é substituído pelo CSO. Os resultados confirmam a hipótese em parte, mostrando que o SYMON exerce grande influência na diminuição do desempenho, porém reforçado pelo cálculo do *fitness*. Os detalhes desta modificação podem ser consultados no Apêndice B.11;

- **DCIAPSO.M8 e DCIAPSO.M9:** verificam a segunda hipótese, levantada após a análise dos resultados obtidos com o DCIAPSO.M6. O SYMON continua sendo substutído pelo CSO na seleção de atributos, como no DCIAPSO.M7. Ao mesmo tempo, o cálculo do *fitness* foi alterado, sendo este o objeto de estudo nestas duas modificações. A conclusão é que as distribuições das classes são complexas ao ponto de tornar inviável o cálculo de *fitness* baseado em distâncias. Os detalhes de ambas as modificações estão presentes no Apêndice B.12;

- **DCIAPSO.M10:** consiste na utilização da métrica AUCarea durante a seleção de atributos. A primeira modificação do DCIAPSO foi similar ao testar a utilização

desta métrica durante a geração dos protótipos. A modificação corrente usa a mesma estratégia, porém antes da execução do DCIA propriamente dito, ou seja, durante o pré-processamento dos dados. Os detalhes desta modificação podem ser consultados no Apêndice B.13. A partir dos resultados é possível concluir que há melhoria de desempenho no modelo proposto;

- **DCIAPSO.M11:** baseado na modificação anterior, utiliza a métrica AUCarea durante a seleção de protótipos com o algoritmo WOA-CM, o qual foi utilizado no DCIAPSO.M4. Apesar de o desempenho ter sido mantido na maioria das bases, em relação ao DCIAPSO.M4, houve queda de desempenho para a base *E.coli*. Mais detalhes podem ser consultados no Apêndice B.14.

3.3.4 Consolidação dos melhores resultados

O DCIAPSO foi utilizado como base em várias propostas que surgiram durante a pesquisa realizada (Seção 3.3.3). A partir dos experimentos foi possível observar que a décima modificação se destacou como a que obteve os melhores resultados. O DCIAPSO.M10 funciona da seguinte forma. Os dados são divididos com $5 - folds$ DOB-SCV, e então normalizados. Em seguida ocorre uma seleção de atributos *wrapper* com o algoritmo de busca CSO (MOAYEDIKIA et al., 2017; GU; CHENG; JIN, 2018). Durante a seleção de atributos, o *fitness* de cada subconjunto é calculado com a métrica AUCarea. Os dados normalizados e *descritos* através de um subconjunto de atributos são então processados pelo DCIAPSO.

Com o DCIAPSO.M10 foi possível observar que técnicas de pré-processamento exercem efeito positivo sobre os dados desbalanceados. Com o pré-processamento realizado foi possível melhorar os resultados obtidos com a geração de protótipos, ao mesmo tempo em que se manteve ou até mesmo houve diminuição da quantidade de protótipos gerados.

A partir disso, foram realizados dois últimos testes, antes das comparações com os algoritmos encontrados na literatura. O primeiro experimento, chamado de DCIASGSA.M5 pode ser visto como a quinta modificação do algoritmo DCIASGSA. O experimento seguinte, chamado DCIACSO é a utilização do CSO durante a geração de protótipos.

O DCIASGSA.M5 implementa todo o pré-processamento realizado sobre os dados no DCIAPSO.M10. A diferença entre esses dois algoritmos se dá no ajuste dos protótipos gerados, desta vez realizado com SGSA. Os resultados são apresentados na Tabela 15.

Em relação ao DCIASGSA, a quinta modificação foi estatisticamente equivalente em quase todas as bases, exceto na base *Thyroid*, onde houve um aumento de desempenho de 0,8792 para 0,9657. Ao se levar em conta também a métrica AUCarea, houve melhoria na base *Car Evaluation*. A comparação com o DCIAPSO.M10 revela que o DCIASGSA.M5 também é estatisticamente equivalente em quase todas as bases, exceto duas, *Gene* e *Zoo*. Na primeira base citada, o DCIAPSO.M10 obteve melhor desempenho, enquanto na

Tabela 15 – Desempenho do DCIASGSA.M5 através das métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8250±0,0129	266,8±62,9500	0,9201
Arrhythmia	0,7506±0,0669	69±8,0000	0,8092
Balance Scale	0,8095±0,0444	6,6±1,5166	0,9868
Car Evaluation	0,9382±0,0237	33,2±3,7014	0,9760
Contraceptive	0,6532±0,0167	19±2,5495	0,9839
Dermatology	0,9341±0,0643	10,2±3,0332	0,9652
E.coli	0,9008±0,0405	30,8±9,3648	0,8854
Gene	0,7664±0,0208	34,2±14,5499	0,9866
Glass	0,8878±0,0466	22±7,5829	0,8715
Hayes-Roth	0,8234±0,0910	11,2±3,7683	0,9125
Horse	0,7144±0,0222	15,8±4,0866	0,9460
Nursery	0,8246±0,0262	15±4,6368	0,9986
Page Blocks	0,8758±0,0201	39,2±7,5631	0,9910
Post Operative	0,5907±0,0310	14±4,4159	0,8056
Satimage	0,9244±0,0114	81±6,5192	0,9843
Shuttle	0,9814±0,0156	61,6±13,2401	0,9987
Soybean	0,9829±0,0063	37,6±6,3482	0,9254
Thyroid	0,9657±0,0187	15±3,5355	0,9974
Wine	0,9906±0,0099	5,4±2,0736	0,9621
Yeast	0,8054±0,0297	77,8±12,5778	0,9345
Zoo	0,9833±0,0136	11,8±3,7014	0,8540

base *Zoo* o DCIASGSA.M5 obteve o melhor desempenho. É interessante notar que uma maneira relativamente pouco diferente de se tratar os dados pode produzir um melhor ou pior desempenho do modelo.

Após verificar o sucesso do DCIASGSA.M5, cujo desempenho é equivalente ao melhor algoritmo implementado até então, houve o teste do DCIACSO. O sucesso do CSO durante a seleção de atributos em relação a outros algoritmos é atrativo. Portanto, foi decidido observar se sua utilização no ajuste dos protótipos gerados pode produzir efeito semelhante, ou seja, uma melhora no desempenho do DCIA.

De forma similar ao DCIASGSA.M5, o DCIACSO pode ser visto como uma alteração no DCIAPSO.M10. Em outras palavras, o DCIACSO executa todo o pré-processamento como descrito no DCIAPSO.M10. A geração de protótipos continua igual, como em todos os algoritmos do DCIA, exceto no ajuste do conjunto de protótipos, no qual foi utilizado o CSO.

Os resultados obtidos com o DCIACSO são apresentados na Tabela 16. Quando comparado ao DCIAPSO.M10 este algoritmo obtem melhor desempenho nas bases *Contra-*

Tabela 16 – Desempenho do DCIACSO através das métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8331±0,0177	260,4±44,9811	0,9220
Arrhythmia	0,7846±0,0402	75,4±14,0107	0,7915
Balance Scale	0,8150±0,0465	6,8±1,6432	0,9864
Car Evaluation	0,9507±0,0135	29,6±10,7145	0,9786
Contraceptive	0,6598±0,0177	14,8±5,4498	0,9874
Dermatology	0,9611±0,0265	9,8±1,0954	0,9665
E.coli	0,9092±0,0311	28±4,0000	0,8958
Gene	0,8242±0,0629	46,6±10,6442	0,9817
Glass	0,8455±0,0251	29,2±7,0143	0,8294
Hayes-Roth	0,8726±0,0346	11±3,8079	0,9141
Horse	0,7324±0,0683	13,6±1,8166	0,9536
Nursery	0,8475±0,0432	14±5,8310	0,9986
Page Blocks	0,8802±0,0171	33,8±7,9812	0,9923
Post Operative	0,6494±0,0825	8,6±4,7749	0,8806
Satimage	0,9253±0,0091	77,2±9,6281	0,9850
Shuttle	0,9880±0,0065	58,6±6,1074	0,9987
Soybean	0,9839±0,0121	41±10,3199	0,9187
Thyroid	0,9715±0,0105	9,2±2,1679	0,9984
Wine	0,9867±0,0085	4,4±0,8944	0,9691
Yeast	0,8005±0,0179	35,8±4,0866	0,9698
Zoo	0,9649±0,0305	8,6±1,1402	0,8936

ceptive e *Shuttle*. Na base *Gene*, apesar de o desempenho ser estatisticamente equivalente, o DCIACSO produz mais protótipos, logo, sendo menos eficiente. Por outro lado, nas bases *Dermatology* e *Post Operative* a quantidade de protótipos gerados é menor. A comparação com o DCIASGSA.M5 revela que ambos os algoritmos possuem desempenhos estatisticamente equivalentes em todas as bases. Entretanto, nas bases *Thyroid* e *Yeast*, o DCIACSO consegue ser mais eficiente ao produzir menos protótipos.

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi relatado todo o processo de desenvolvimento e análises realizadas. A fim de se abordar o problema de múltiplas classes desbalanceadas de forma simples, porém eficiente, optou-se por verificar uma abordagem baseada em geração de protótipos.

A primeira ideia consistiu em dividir o espaço amostral em diversas pequenas células, cada qual com seu respectivo centroide. Com tal divisão se esperava um mapeamento bastante preciso das classes presentes em uma determinada base de dados. Entretanto, esta

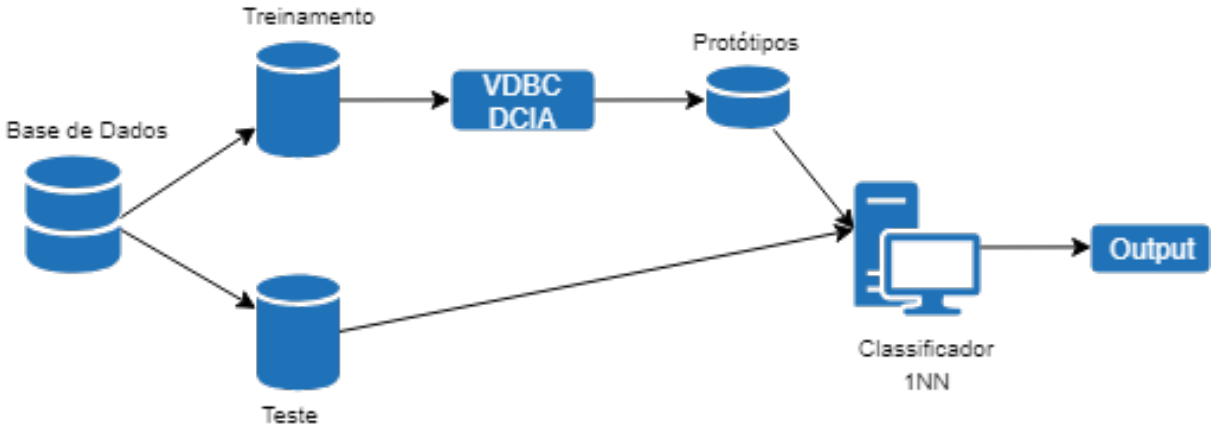

Figura 26 – Esquema geral do modelo de classificação desenvolvido.

abordagem pode ter dois grandes problemas. O primeiro consiste no custo computacional, dependendo do quanto se divida o espaço. O segundo problema, consiste no possível *overfitting*, após um mapeamento bastante preciso das classes, ao se observar somente o conjunto de treinamento.

Como primeira medida tomada foi implementado e testado o algoritmo VDBC, baseado no algoritmo W* de Chang (CHANG, 1974). Os resultados obtidos foram entre razoáveis e bons, ou seja, com o desempenho médio em cerca de 0,8. Além disso houve uma redução de instâncias razoável, atingido em média 50% do tamanho original do conjunto de treinamento. Sobre este algoritmo foram testadas cinco modificações a fim de se observar a variação no desempenho. As conclusões sobre esses estudos estão escritos na Seção 3.2.1.

Dentre as conclusões, houve duas que chamaram atenção e serviram de motivação para o DCIA, o próximo algoritmo a ser desenvolvido. Foi observado com o VDBC que a criação de alguns protótipos a partir de protótipos existentes, que na prática resulta na *movimentação* dos mesmos pelo espaço, exerce influência sobre o desempenho do algoritmo. Além disso, foi possível observar também que é possível obter um bom desempenho com uma quantidade bastante reduzida de protótipos gerados.

O DCIA foi então desenvolvido baseando-se no algoritmo IPADE-ID (LÓPEZ et al., 2014), de forma que os protótipos não somente são gerados como também suas posições passam a ser ajustadas. No VDBC, o conjunto de treinamento final era construído a partir de uma redução do conjunto de treinamento inicial, de forma que o procedimento geral é similar à técnica de seleção de protótipos. No DCIA, o novo conjunto de treinamento é construído a partir de um protótipo por cada classe. Essas alterações resultaram em um bom desempenho de classificação ao mesmo tempo em que foi possível uma maior redução de instâncias.

Foram realizados vários experimentos utilizando o DCIA, levando-se em conta também várias modificações. Os principais algoritmos nesta fase foram DCIAGSA, DCIASGSA,

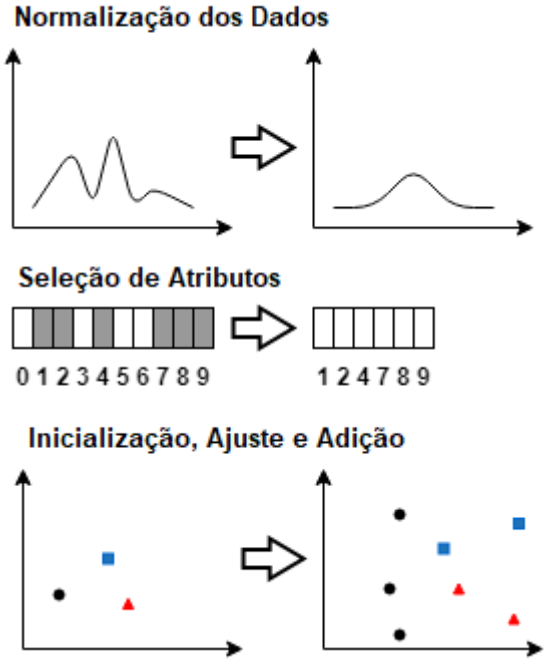

Figura 27 – Esquema geral do DCIA.

DCIAPSO e DCIACSO. Sobre cada um desses algoritmos, exceto o último, ao menos uma modificação foi testada. Os algoritmos que obtiveram os melhores resultados foram DCIASGSA.M5, DCIAPSO.M10 e DCIACSO.

A Figura 26 apresenta um esquema geral do modelo de classificação desenvolvido neste trabalho, enquanto a Figura 27 apresenta o modelo geral do DCIA. De forma resumida, o modelo de classificação consiste em dividir os dados em dois subconjuntos, gerar um conjunto de protótipos a partir do subconjunto de treinamento e testar os dados do subconjunto de teste utilizando o classificador 1NN, o qual utiliza os protótipos gerados para classificar as instâncias desconhecidas. Este trabalho focou em duas partes deste esquema: (1) pré-processamento e (2) geração de protótipos. No pré-processamento se destacaram a normalização dos dados e a seleção de atributos. Na geração de protótipos, modificações no DCIA foram as mais bem sucedidas.

Com os experimentos realizados com o DCIA, algumas conclusões podem ser destacadas:

- A geração de protótipos iniciada com um conjunto pequeno, a ser incrementado é mais eficiente em reduzir o tamanho do conjunto de treinamento;
- A normalização dos dados exerce efeito bastante positivo sobre a classificação em bases com múltiplas classes desbalanceadas:
  - Este efeito ocorre principalmente em classes cujos valores de atributos variam bastante e/ou possuem classes que são bastante próximas umas das outras;

- A divisão dos dados com a técnica DOB-SCV não surtiu efeito na classificação das bases com múltiplas classes desbalanceadas. Há duas possíveis razões para tal:
  - As bases utilizadas não sofrem de *dataset-shift* ou,
  - A divisão dos dados em *5-fold* não é suficiente para a ocorrência do problema;
- Sobre o DCIA, tanto sobreamostragem quanto subamostragem não resultam em melhoria de desempenho de classificação;
- A geração de protótipos com o DCIA consegue "ignorar" a complexidade de uma base de dados, pois a quantidade de protótipos gerados independe das taxas de desbalanceamento (IR e MIR). Por outro lado, o VDBC gera mais protótipos caso o desbalanceamento seja maior;
- Características das bases, como maior separabilidade entre suas classes e uma maior quantidade de atributos para seleção, podem ter influência positiva sobre o desempenho de classificação;
- A métrica AUCarea consegue influenciar positivamente no desempenho de classificação quando utilizada na seleção de atributos;
- A seleção de atributos do tipo *wrapper* demonstrou ser uma boa prática na classificação de bases com múltiplas classes desbalanceadas:
  - Contudo, nem todos os algoritmos de seleção de atributos, mesmo os utilizados para bases desbalanceadas binárias, terão bons desempenhos garantidos na presença de três ou mais classes desbalanceadas;
  - Dos algoritmos de busca testados, o CSO produziu o melhor desempenho;
- Utilizar métricas diferentes durante o modelo de classificação pode resultar em melhoria de desempenho;
- A complexidade da disposição das classes no espaço amostral faz com que a seleção de atributos considerando as distâncias entre as instâncias não seja uma boa prática;
- De forma similar ao VDBC, foi observado que algumas modificações afetam o desempenho dos classificadores em algumas bases específicas, seja positiva ou negativamente. Por exemplo, em algumas situações houve melhoria de desempenho com aumento de protótipos gerados, ou o contrário. Esta ocorrência sugere que adaptações *ad hoc* no algoritmo possam ser uma alternativa válida.

Após os vários experimentos conduzidos, os quais permitiram diversas análises anteriormente pontuadas, ainda se faz necessário as comparações do algoritmo desenvolvido com algoritmos encontrados na literatura e abordam o mesmo problema. Desta forma, será

possível observar a qualidade dos resultados produzidos pelo DCIA. Também será possível verificar se, com a simplicidade empregada, é possível atingir ou superar o desempenho de classificação de outros algoritmos mais complexos.

4 COMPARAÇÕES COM ALGORITMOS DA LITERATURA

No Capítulo 3 foi apresentado o método proposto e sua evolução, através de um conjunto de variações e algoritmos complementares. Os melhores desempenhos foram alcançados com os algoritmos DCIACSO, DCIAPSO.M10 e DCIASGSA.M5[1]. Entretanto, o DCIA ainda necessita ser validado em relação à literatura existente sobre o tema. Por isso, neste capítulo, o DCIA é comparado a outros algoritmos.

As comparações são separadas em três abordagens: (1) algoritmos em nível de dados simples e amplamente conhecidos, (2) algoritmos em nível de dados mais complexos ou *estado-da-arte* e, (3) soluções em nível de algoritmo, também *estado-da-arte*. O objetivo é verificar o desempenho do DCIA em relação a demais algoritmos similares, i.e., que pré-processam os dados. Além disso, outro objetivo é verificar seu desempenho em relação a outros algoritmos que atacam o mesmo problema, contudo de diferentes maneiras.

Na primeira abordagem, o DCIA, com suas três melhores variações, é comparado aos algoritmos ADASYN (HE et al., 2008), *Borderline* SMOTE (HAN; WANG; MAO, 2005), ROS, RUS, *Safe Level* SMOTE (BUNKHUMPORNPAT; SINAPIROMSARAN; LURSINSAP, 2009) e *Static* SMOTE (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011).

Na segunda abordagem as comparações são com os algoritmos *estado-da-arte* e em nível de dados: DSRBF (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011), MDO (ABDI; HASHEMI, 2016), MDO+ e AMDO (YANG et al., 2018). Por fim, na terceira abordagem as comparações são com as soluções *estado-da-arte* em nível de algoritmo: AdaBoost.NC (WANG; YAO, 2012), AdaC2.M1 (SUN; KAMEL; WANG, 2006), AMCS (YI-JING et al., 2016), CoMBo (KOÇO; CAPPONI, 2013) e DECOC (BI; ZHANG, 2018).

As comparações com os algoritmos citados vieram acompanhados de algumas dificuldades. Uma vez que os estudos sobre bases de dados desbalanceadas são relativamente recentes, principalmente no cenário de múltiplas classes, não existe ainda uma padronização nos experimentos.

Por exemplo, quando alguma base possui classes com menos de dez instâncias, alguns autores preferem ignorar estas classes raras. Por outro lado, outros autores utilizam todas as classes, mesmo as que possuem somente uma instância. Neste caso, a instância é replicada para que a mesma sempre apareça no conjunto de treinamento. Com a técnica de separação de dados *5-fold*, uma classe com duas instâncias estará presente igualmente no conjunto de treinamento e teste em duas configurações, enquanto nas três configurações restantes estará presente somente no conjunto de treinamento. No caso de uma classe cujo *singleton* foi replicado, algum pesquisador poderá forçar a presença da classe nos conjuntos de treinamento e teste em todas as configurações formadas pelo *k-fold*. Entretanto,

[1] Os dois últimos algoritmos citados serão referidos neste capítulo como DCIAPSO e DCIASGSA, respectivamente.

surgem as perguntas: como deveria ser o procedimento experimental para problemas com classes com duas ou três instâncias? Qual instância deve ser replicada ou forçada a aparecer no conjunto de teste? Ou seria melhor a utilização de sobreamostragem nessas classes extremamente raras?

Infelizmente não foi encontrado um estudo que dê atenção ou responda completamente a essas perguntas. Na maioria dos trabalhos os autores não deixam claro qual abordagem utilizaram. Ainda, apesar de ser possível a ausência de uma classe no conjunto de teste, é interessante que todas as classes estejam presentes nos conjuntos de treinamento e teste. Caso alguma classe esteja ausente do conjunto de teste, o cálculo de algumas métricas pode ficar prejudicado devido a multiplicações ou divisões pelo valor 0. Por fim, um breve estudo (SILVA; ZANCHETTIN, 2016a) sugere a utilização de *Repeated Hold-Out* 33% para bases de dados com classes extremamente raras.

A validação dos experimentos, entretanto, é mais comumente realizada com a técnica *k-fold*. Contudo, não existe ainda uma padronização do valor k em experimentos com bases desbalanceadas. Dividir os dados em 10 *folds* é bastante comum na literatura, sendo até normalmente o recomendado (KOHAVI, 1995). Todavia, classes raras estão presentes em várias das bases de dados comumente utilizadas. Devido a isto, muitos autores têm preferido a utilização de *5-folds*, pois há maiores chances de todas as classes estarem presentes em todos os *folds*. Esta configuração de validação experimental aparentemente está se tornando padrão em estudos com bases desbalanceadas, mas ainda é possível encontrar diversas outras configurações, como *10-folds*, às vezes com repetição, *hold-out* e repetições de *5-folds*.

Outra particularidade que dificulta as comparações é o pré-processamento de instâncias com *missing values*, ou valores inexistentes para determinado atributo. Vários autores não deixam claro que abordagem seguiram, desta forma, torna-se mais difícil saber se as bases de dados, ainda que possuam mesmo nome e fonte, sejam de fato idênticas. Em vários casos é possível constatar que existem diferenças ao se observar a quantidade de instâncias ou a quantidade de atributos. Outra informação relevante é a quantidade de instâncias por classe, a qual pode ajudar na identificação de bases pré-processadas. Contudo, além de não informarem se pré-processaram alguma base de dados, alguns autores omitem também algumas dessas informações.

Uma maneira de se contornar esse problema é com o fornecimento de um código-fonte do algoritmo e do experimento. No arquivo do código-fonte podem estar presentes as bases utilizadas, uma função para o pré-processamento, ou mesmo o algoritmo pronto para receber qualquer base de dados. Infelizmente, ter acesso aos códigos-fonte também é outro problema. Muitos dos algoritmos, principalmente os mais sofisticados, podem ter suas descrições um pouco confusas, ou ainda algum detalhe que possa ser relevante para um melhor funcionamento do algoritmo ter sido excluído. Ademais, muitos autores demoram bastante a responder solicitações de informações experimentais, ou mesmo não

respondem às tentativas de contato, algumas vezes por não utilizarem mais o meio de contato fornecido.

Reimplementar tais algoritmos pode ser uma solução. Contudo, o tempo necessário para o correto entendimento de todos os passos e a implementação em si, pode exigir um tempo não disponível ao pesquisador. A solução para este caso pode ser simular o desempenho dos algoritmos criando uma distribuição com as médias e desvios-padrões encontrados na literatura. Porém, em vários casos, tais resultados são dispostos através de uma variedade de métricas diferentes. O pior cenário se dá quando somente a média é publicada, ou os resultados são publicados utilizando ranqueamentos de desempenho.

Devido às várias das dificuldades relatadas, para que houvesse comparações estatisticamente válidas, o DCIA foi executado novamente, sempre em adaptação às diversas configurações de experimentos encontradas. Como exemplo, novas bases de dados foram adicionadas, algumas modificadas, e outras métricas foram também calculadas. Portanto, todas as adaptações implementadas, como também a descrição das novas bases utilizadas e outros detalhes inerentes a cada comparação, são detalhados em suas respectivas seções.

Por fim, para que todas as dificuldades relatadas, ou pelo menos sua maior parte, sejam mitigadas em relação a esta pesquisa, o código-fonte, tanto do VDBC, quanto do DCIA estão disponíveis integralmente em meio eletrônico. O VDBC pode ser encontrado no endereço <https://github.com/EvandroJRSilva/VDBC>, enquanto o DCIA pode ser encontrado em <https://github.com/EvandroJRSilva/DCIA>.

4.1 PRIMEIRA ABORDAGEM

Algoritmos em nível de dados pré-processam as instâncias de treinamento com o objetivo de entregarem a algum classificador um conjunto de dados que possa ser melhor modelado visando a generalização do modelo. O DCIA, por exemplo, gera protótipos que são utilizados pelo classificador 1NN. Entretanto, é possível que os protótipos gerados sejam utilizados por qualquer outro classificador, seja ele supervisionado ou não.

O objetivo das comparações nessa primeira abordagem é comparar o desempenho de vários classificadores ao aprenderem a distribuição dos dados do problema com um conjunto de treinamento pré-processado ou não. Os classificadores, ou modelos de classificação, utilizados são: 1NN, 5NN, AdaBoost.M2 (FREUND; SCHAPIRE, 1997), CART (BREIMAN et al., 1984), *Error Correcting Output Codes* (ECOC) (DIETTERICH; BAKIRI, 1991) com diferentes classificadores base e *Random Forest* (BREIMAN, 2001).

Os algoritmos para pré-processamento dos conjuntos de treinamento são aqueles citados no início deste capítulo. Entretanto, exceto pelo *Static* SMOTE (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011), todos os demais foram construídos originalmente para o tratamento de bases binárias. O *Static* SMOTE é uma adaptação do algoritmo original para o contexto de bases de dados com múltiplas classes. Seu funcionamento é da seguinte forma. O processo de sobreamostragem acontece em C passos,

Tabela 17 – Exemplo de sobreamostragem com *Static* SMOTE na base *Glass*.

Iteração	**Número de instâncias por classe**	**Classe selecionada**
1	(70, 76, 17, 13, 9, 29)	5
2	(70, 76, 17, 13, 18, 29)	4
3	(70, 76, 17, 26, 18, 29)	3
4	(70, 76, 34, 26, 18, 29)	5
5	(70, 76, 34, 26, 27, 29)	4
6	(70, 76, 34, 39, 27, 29)	5
-	(70, 76, 34, 39, 36, 29)	-

onde C é o número de classes. Em cada passo a menor classe é encontrada e a quantidade original de suas instâncias é duplicada com SMOTE. Entretanto, para que somente as classes menores passem pela sobreamostragem, a seguinte condição proposta para o *Static* SMOTE no DSRBF também foi utilizada neste trabalho:

$$p^* \leq \frac{1}{2 \cdot C} \tag{4.1}$$

no qual p^* é a menor probabilidade *a priori* entre as classes. A Tabela 17 apresenta um exemplo adaptado retirado do artigo que propõe o algoritmo. Neste trabalho, os demais algoritmos de sobreamostragem, i.e., ADASYN, *Borderline* SMOTE e *Safe Level* SMOTE também foram todos adaptados para bases com múltiplas classes da mesma forma que o *Static SMOTE*.

A adaptação do ROS segue a seguinte heurística: a quantidade de instâncias da maior classe $maxSize$ é encontrada. Para cada classe c, a diferença de tamanho $maxSize - size(c)$ é calculada. Este valor é a quantidade de novas instâncias que a classe corrente deve possuir. Portanto, instâncias aleatórias da classe são duplicadas até que a mesma atinja o tamanho da maior classe.

Por fim, o algoritmo RUS foi adaptado como segue. A quantidade total de instâncias é dividida pela quantidade de classes. Este valor é a quantidade de instâncias que cada classe deveria possuir para que a base fosse balanceada. Então, para cada classe, caso sua quantidade de instâncias seja superior ao valor calculado, instâncias aleatórias são selecionadas e excluídas até que a mesma chegue ao tamanho desejado.

Todos esses algoritmos foram executados sobre trinta e uma bases de dados retiradas dos repositórios UCI (DHEERU; TANISKIDOU, 2017) e KEEL (ALCALA-FDEZ et al., 2010). Os mesmos são descritos na Tabela 18. É interessante notar que algumas das bases não são desbalanceadas, ou praticamente balanceadas (e.g., *Segment*, *Vehicle* e *Vowel*). As mesmas foram acrescentadas por estarem presentes em outros estudos da área.

Os experimentos foram conduzidos com *5-fold stratified cross-validation*. Todos os

Tabela 18 – Resumo das bases de dados usadas nos testes na Primeira Abordagem.

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Abalone	23	8	4172	344,5	268,5644
Autos	6	25	159	16	8,2388
Balance Scale	3	4	625	5,8776	2,6985
Car Evaluation	4	6	1728	18,6154	10,3890
Cleveland	5	13	303	12,6154	4,5476
Contraceptive	3	9	1473	1,8889	0,2159
Dermatology	6	34	366	5,6	1,8598
E.coli	8	7	336	71,5	47,3469
Flare	6	11	1066	7,6977	3,3327
Gene	3	60	3190	2,1578	0,4134
Glass	6	9	214	8,4444	5,0133
Hayes-Roth	3	4	160	2,0968	0,3743
Horse	3	14	366	4,3269	1,2592
Landsat	6	36	2000	2,2275	0,7463
Led7Digit	10	7	500	1,5405	0,1480
Lymphography	4	18	148	40,5	24,8133
New Thyroid	3	5	215	1,6667	1,9143
Nursery	5	8	12960	2160	1300,8
Page Blocks	5	10	5473	175,4643	59,5996
Penbased	10	16	1100	1,0952	0,0162
Post Operative	3	8	90	32	13,7188
Satimage	6	36	6435	2,4489	0,9564
Segment	7	19	2310	1	0
Soybean	15	35	630	4,6	7,6660
Thyroid	3	21	7200	40,1566	18,3396
Vehicle	4	18	846	1,0955	0,0053
Vowel	11	13	990	1	0
Wine	3	13	178	1,4792	0,0774
Wine QR	6	11	1599	68,1	42.6321
Yeast	10	8	1484	92,6	44,7543
Zoo	7	16	101	10,25	4,9225

algoritmos foram implementados em Matlab, versão 2018a, em máquinas diferentes. Os classificadores e modelos de classificação utilizados estão nativamente implementados no mesmo *software*. Para o AdaBoost.M2 a taxa de aprendizado é 0,1. O método ECOC foi utilizado com três diferentes classificadores, ou modelos de classificação: AdaBoost.M1, CART e SVM. Sobre o primeiro, a taxa de aprendizado foi configurada com o valor 0,1, o classificador base é o CART, e a quantidade de classificadores foi configurada com o

valor 100. Quanto ao SVM, foram testados dois kernels, *linear* e *RBF*. Todas as demais configurações permaneceram as padrões do modelo implementado no *toolbox* do Matlab. A métrica utilizada para análise é a MAUC (HAND; TILL, 2001).

A quantidade de algoritmos a serem comparados faz com que testes entre pares fiquem impraticáveis. Desta forma, os testes estatísticos realizados foram o ANOVA (ARMSTRONG; S.V.SLADE; F.EPERJESI, 2000), caso todos os resultados tenham uma distribuição normal, ou Kruskal Wallis (KRUSKAL; WALLIS, 1952; KRUSKAL-WALLIS..., 2008), caso contrário. Para facilitar da visualização dos resultados dos experimentos se optou por utilizar também o *Critical Difference Diagram* (CDD) (DEMŠAR, 2006) com 5% de nível de significância. O gráfico do CDD mostra em uma linha numerada a média aritmética dos ranks obtidos pelos diferentes classificadores. Quanto menor o rank, ou seja, mais próximo do valor 1, melhor. Caso as médias difiram em pelo menos o valor do *Critical Difference* (CD), ou diferença crítica, os mesmos podem ser considerados estatisticamente diferentes. Caso as médias sejam estatisticamente equivalentes, as mesmas são conectadas por uma linha horizontal.

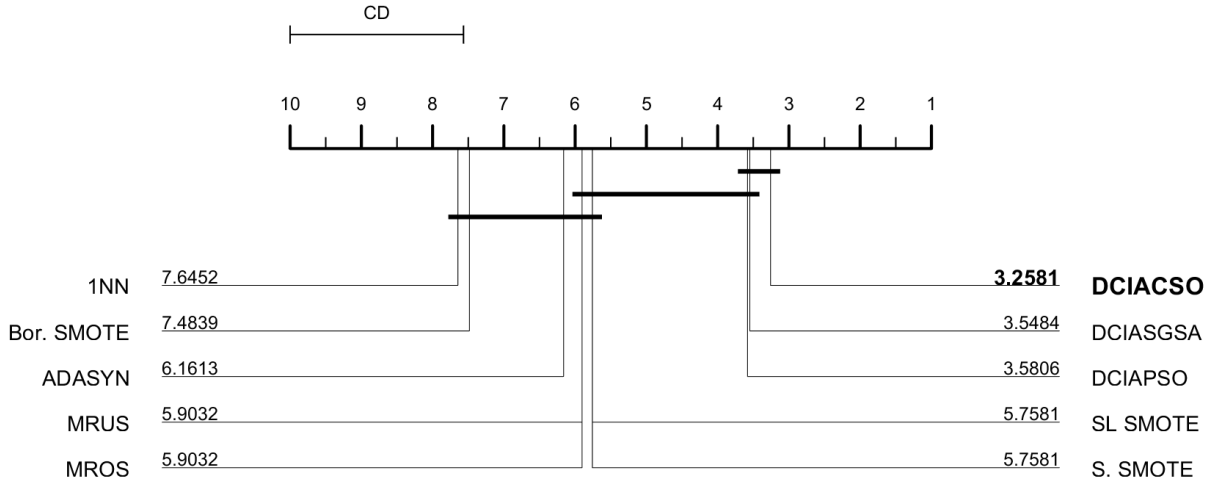

Figura 28 – CDD do desempenho obtido pelo classificador 1NN com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

A partir da Figura 28 é possível perceber que, em se tratando do classificador 1NN o DCIA consegue o melhor desempenho, principalmente utilizando o CSO. Os demais algoritmos, apesar de terem conseguido melhores ranks, possuem desempenho estatisticamente equivalente ao próprio 1NN, ou seja, o classificador executado sobre o conjunto de treinamento sem qualquer pré-processamento. O sucesso do DCIA neste caso pode ser explicado pelo fato de o mesmo gerar protótipos associados ao desempenho do 1NN durante sua execução.

Na Figura 29 são apresentados os ranks obtidos pelos algoritmos com o classificador 5NN. É possível perceber que o ROS adaptado para múltiplas classes (MROS) juntamente com o *Static* SMOTE (S. SMOTE) obtiveram os melhores desempenhos. Ainda, cinco dos

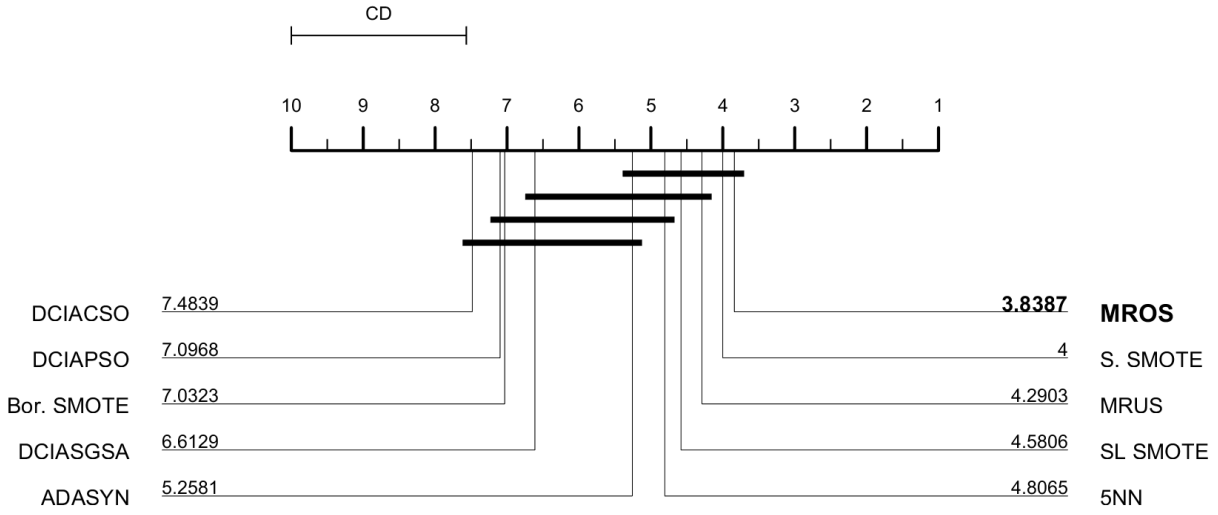

Figura 29 – CDD do desempenho obtido pelo classificador 5NN com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

nove algoritmos foram piores que o próprio 5NN, levando-se em conta a média dos ranks. Desses cinco algoritmos, o DCIACSO é o único que pode ser considerado estatisticamente inferior ao 5NN.

Existem duas explicações para o pior desempenho do DCIA com o 5NN. A primeira é que a geração de protótipos só considera os desempenhos obtidos com o 1NN. Pode ser possível que a geração de protótipos considerando o 5NN resulte em melhora de desempenho. Entretanto, é necessário um estudo posterior para confirmação. A segunda razão para a degradação do desempenho é o fato de a quantidade de protótipos gerados ser insuficiente para uma boa generalização do classificador. Por exemplo, em algumas bases de dados são gerados menos de cinco protótipos, ou pouco mais de um protótipo para cada classe. Uma instância desconhecida, então, tem de ser classificada levando em conta cinco ou menos vizinhos, todos de classes diferentes. Caso a geração de protótipos seja executada com o 5NN, existe a possibilidade de que a quantidade de protótipos gerados ser maior e, portanto, o desempenho também seja melhorado.

Os resultados dos algoritmos com o AdaBoost.M2 podem ser visualizados na Figura 30. O MROS outra vez se destacou, enquanto o DCIA ficou entre os de baixo desempenho. A explicação para o baixo desempenho obtido com o DCIA é que o AdaBoost é prejudicado quando o conjunto de treinamento é pequeno (LI et al., 2004). Esta é também a explicação para que o RUS adaptado para múltiplas classes (MRUS) também tenha obtido desempenho pior. O *Borderline* SMOTE (Bor. SMOTE) pode ter sofrido um pouco com bases pequenas. Além disso, o fato de focar na criação de instâncias próximas às bordas entre as classes, ou seja, instâncias que normalmente são mais difíceis de classificar, também pode ter prejudicado seu desempenho.

Levando-se em conta o classificador CART (Figura 31), é possível notar que a quantidade mínima de instâncias para treinamento fornecidas pelo DCIA também é prejudicial

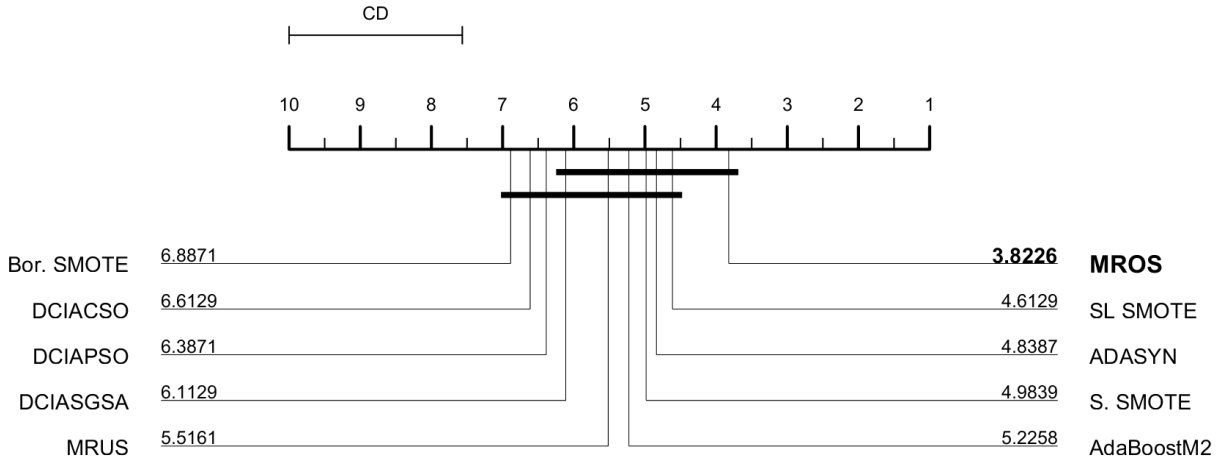

Figura 30 – CDD do desempenho obtido pelo AdaBoost.M2 com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

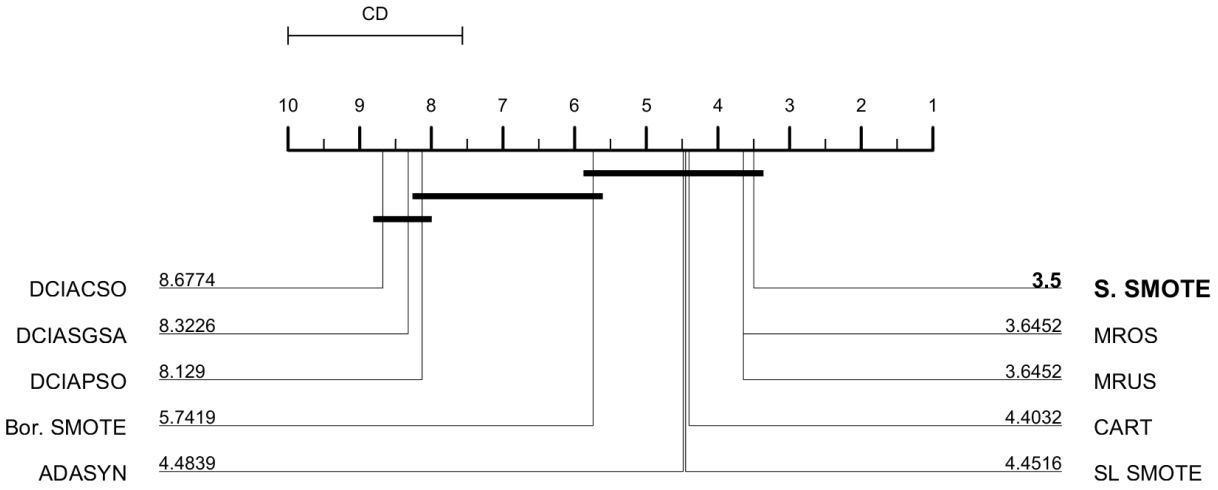

Figura 31 – CDD do desempenho obtido pelo classificador CART com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

para o seu desempenho. É interessante notar também o bom posicionamento das técnicas mais simples, inclusive na ausência de pré-processamento. As médias dos ranks sugerem que para a árvore de decisão, técnicas simples de *resampling*, de preferência sobreamostragem, são suficientes para um bom ganho de desempenho.

O ECOC pode ser entendido como um método de decomposição OVA. Desta forma é possível utilizar o AdaBoost.M1 para classificações binárias. A Figura 32 apresenta o CDD dos ranks obtidos com essa configuração. O que se pode notar é que todos os algoritmos são estatisticamente equivalentes. Cada classe foi aprendida por um comitê de classificadores (AdaBoost.M1), os quais foram agrupados em outro comitê maior (ECOC). Desta forma, independente do algoritmo aplicado, os classificadores e a agregação dos mesmos resultaram em desempenhos similares.

O desempenho obtido pelo classificador CART com o método ECOC (Figura 33), é similar ao seu desempenho sem a utilização de binarização. A decomposição, contudo,

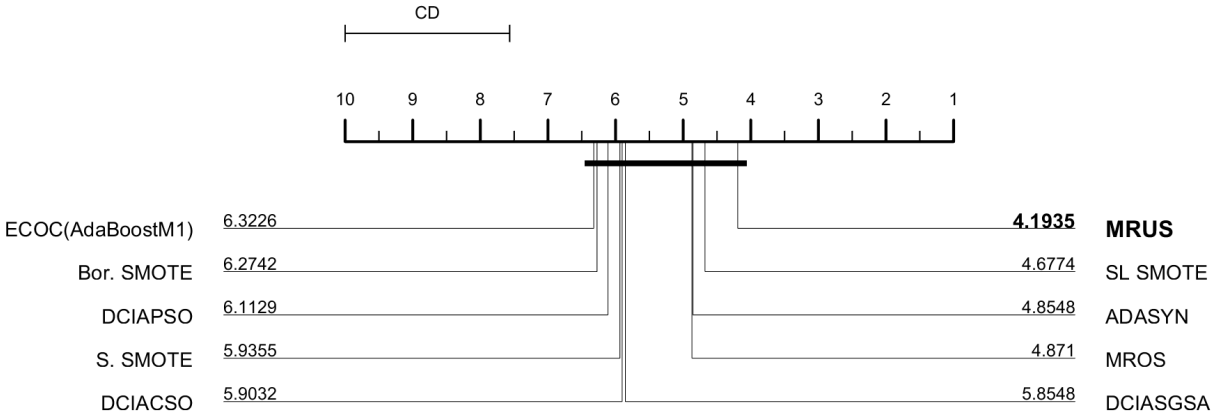

Figura 32 – CDD do desempenho obtido pelo AdaBoost.M1 através do método ECOC com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

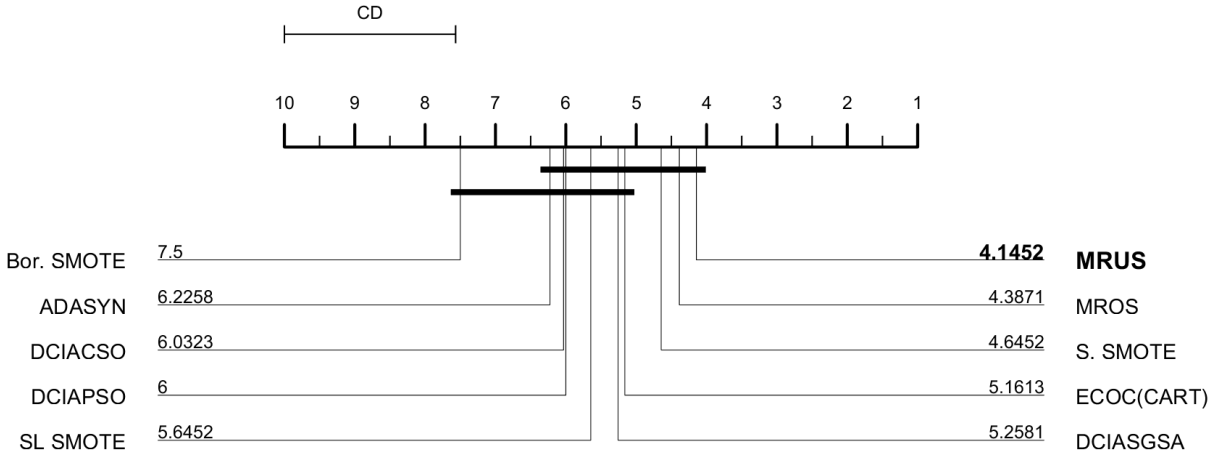

Figura 33 – CDD do desempenho obtido pelo classificador CART através do método ECOC com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

possibilitou que a maioria dos algoritmos sejam estatisticamente equivalentes, exceto o Bor. SMOTE.

Os resultados das comparações com a utilização do SVM no método ECOC podem ser visualizados nas Figuras 34 e 35. Enquanto os resultados do SVM com kernel linear seguem a tendência observada nos demais classificadores, o SVM com kernel RBF obtém seu melhor desempenho com o DCIA. Isto se dá pelo fato de ambos estarem atrelados às distâncias entre as instâncias. Os protótipos gerados são responsáveis por áreas do espaço amostral, de forma que toda instância dentro de seu raio de influência é classificada de acordo com sua classe.

As últimas comparações podem ser visualizadas na Figura 36. Nesta figura, foi possível perceber que os métodos mais simples, como MROS e MRUS, costumam influenciar positivamente no desempenho do modelo quando existe um comitê de classificadores. Resultados similares podem ser observados também nos resultados com AdaBoost.M2 e

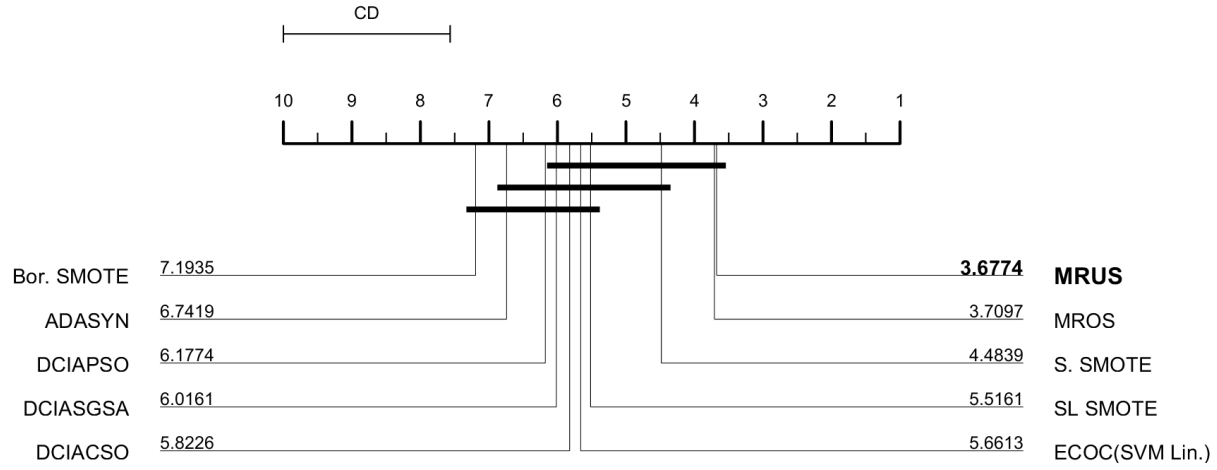

Figura 34 – CDD do desempenho obtido pelo classificador SVM com kernel linear através do método ECOC com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

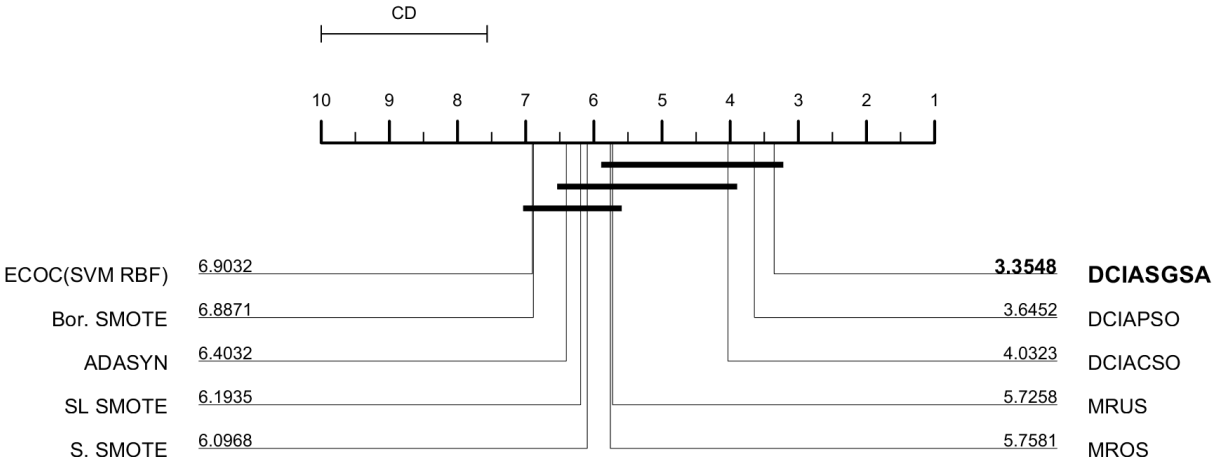

Figura 35 – CDD do desempenho obtido pelo classificador SVM com kernel RBF através do método ECOC com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

ECOC.

A partir da análise dos experimentos é possível delinear as seguintes conclusões:

- O DCIA se destaca quando a classificação depende da distância entre as instâncias, e.g., com os classificadores 1NN e SVM com kernel RBF;
- Um balanceamento simples, efetuado pelo MROS ou MRUS é suficiente para ganho de desempenho de boa parte dos classificadores testados;
- Em todos os casos observados, a classificação sem pré-processamento do conjunto de treinamento resulta em desempenho médio estatisticamente equivalente à maioria dos demais algoritmos;

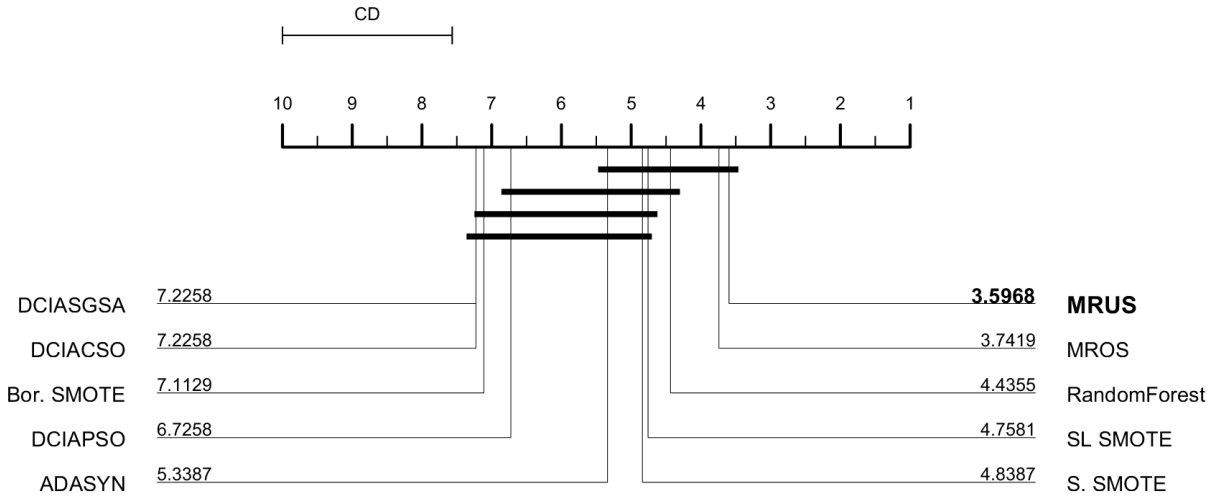

Figura 36 – CDD do desempenho obtido pelo comitê de classificadores Random Forest com e sem o conjunto de treinamento pré-processado pelos respectivos algoritmos.

- A alta taxa de redução do conjunto de treinamento efetuado pelo DCIA é a principal razão deste algoritmo ocasionar um menor desempenho na maioria dos classificadores. A utilização de outros classificadores durante o processo de geração de protótipos para melhora de desempenho é uma hipótese ainda a ser estudada;

- Se observa que nenhum algoritmo de reamostragem se destacou ao influenciar um ganho de desempenho para todos os classificadores. Desta forma, pode ser interessante a busca de algum padrão que ajude na escolha automática da combinação entre algoritmo de pré-processamento e classificador para o alcance do melhor resultado.

Por fim, é importante frisar que o CDD é construído sobre as médias de desempenho dos algoritmos em uma série de bases de dados. O ranqueamento não levou em conta o desvio-padrão das médias. Por exemplo, se algum algoritmo tenha obtido 0,9 de MAUC, seu rank será melhor que o de um algoritmo que tenha obtido 0,89 na mesma métrica. Uma análise exaustiva dos resultados levando em conta os desvios-padrões em cada base de dados pode levar a conclusões um pouco diferentes.

Entretanto, não há razões para acreditar que as diferenças entre as análises venham a ser significativas. Caso as médias de dois algoritmos sejam sempre próximas, e, por muitas vezes considerados estatisticamente equivalentes devido aos seus desvios-padrões, espera-se que, com o cálculo do CD, as médias de seus ranks também sejam estatisticamente equivalentes.

4.2 SEGUNDA ABORDAGEM

As próximas comparações realizadas ainda levam em conta algoritmos em nível de dados, entretanto estes podem ser considerados mais complexos ou *estado-da-arte* na litera-

Tabela 19 – Resumo das bases de dados usadas pelo algoritmo DSRBF.

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Anneal	5	59	898	85,5	26,6973
Balance Scale	3	4	625	5,8776	2,6985
E.coli	8	7	336	71,5	47,3469
Glass	6	9	214	8,4444	5,0133
Saureus4	4	3	287	9,75	4,8218
New Thyroid	3	5	215	5	1,9143
Yeast	10	8	1484	92,6	44,7543
Zoo	7	16	101	10,25	4,9225

Tabela 20 – Desempenhos do DSRBF, DCIACSO, DCIAPSO e DCIASGSA utilizando a média e desvio-padrão da métrica Acurácia

Base	**DSRBF**	**DCIACSO**	**DCIAPSO**	**DCIASGSA**
Anneal	**0,9834**±0,0197	0,9344±0,0292	0,9065±0,0359	0,9157±0,0365
Balance Scale	**0,9349**±0,0315	0,7885±0,0493	0,7852±0,1333	0,7769±0,0365
E.coli	**0,8600**±0,0736	0,7659±0,0780	0,7631±0,0701	0,7676±0,0650
Glass	**0,6525**±0,0897	0,5798±0,1280	0,5886±0,1263	0,5727±0,1090
Saureus4	**0,7320**±0,0346	0,5443±0,0803	0,5276±0,1155	0,5414±0,1104
New Thyroid	**0,9675**±0,0238	0,9320±0,1095	0,9181±0,1411	0,9321±0,0883
Yeast	**0,5869**±0,0447	0,4198±0,0485	0,3658±0,0837	0,3291±0,1054
Zoo	0,9375±0,0532	0,9173±0,0738	0,9188±0,0777	0,9170±0,0678

tura observada. São quatro os algoritmos a serem comparados ao DCIA: DSRBF, MDO, MDO+ e AMDO.

Para esta seção, as comparações foram realizadas entre pares de algoritmos, utilizando a mesma configuração e métrica apresentadas nos trabalhos originais comparados. Para o DSRBF (FERNÁNDEZ-NAVARRO; HERVÁS-MARTÍNEZ; GUTIÉRREZ, 2011) a configuração é de *10-fold* com dez repetições por *fold*, e as métricas utilizadas são Acurácia e Sensibilidade Mínima. O problema da Acurácia em bases binárias não é ubíquo em bases com múltiplas classes. Entretanto, ainda continua não sendo uma boa métrica. A Sensibilidade Mínima (SM), por outro lado, faz mais sentido em bases binárias, no qual existe grande destaque na generalização da classe minoritária. Tais escolhas dos autores podem ser entendidas em função da data de publicação, ou seja, 2011. Neste período este trabalho poderia ser visto como pioneiro em múltiplas classes desbalanceadas.

As médias e desvios-padrões do DSRBF foram extraídos do trabalho original. A partir dessas informações foram criadas distribuições normais para comparações. Ou seja, os valores apresentados nesta tese não necessariamente são idênticos aos apresentados

Tabela 21 – Desempenhos do DSRBF, DCIACSO, DCIAPSO e DCIASGSA utilizando a média e desvio-padrão da métrica Sensibilidade Mínima

Base	**DSRBF**	**DCIACSO**	**DCIAPSO**	**DCIASGSA**
Anneal	0,7491±0,1805	0,7374±0,3207	0,7162±0,3213	0,7481±0,2993
Balance Scale	**0,7748**±0,1512	0,6929±0,1297	0,2496±0,2840	0,7035±0,0823
E.coli	**0,3754**±0,2742	0,1396±0,2132	0,1295±0,2210	0,1254±0,2192
Glass	**0,1801**±0,1826	0,0762±0,1603	0,0709±0,1728	0,0723±0,1616
Saureus4	**0,2048**±0,0948	0,0846±0,1708	0,0917±0,1727	0,1053±0,1811
New Thyroid	0,9059±0,0890	0,8613±0,1685	0,8418±0,1932	0,8182±0,2162
Yeast	**0,0773**±0,0718	0,0009±0,0053	0,0002±0,0021	0,0006±0,0037
Zoo	**0,6080**±0,3318	0,3650±0,4812	0,3700±0,4852	0,3450±0,4738

no trabalho original. Quanto ao DCIA, o mesmo foi novamente executado seguindo a configuração dos experimentos do DSRBF.

A Tabela 20 apresenta os desempenhos obtidos pelos quatro algoritmos através da métrica Acurácia. De forma complementar, a Tabela 21 apresenta os desempenhos utilizando a métrica Sensibilidade Mínima. As comparações são feitas entre pares DSRBF vs. DCIA(...). Os melhores desempenhos, estatisticamente, são destacados em **negrito**. Como as comparações são entre pares, o DSRBF só é destacado quando supera todas as versões do DCIA. Os testes foram executados com *Student's t-test* (SNEDECOR; COCHRAN, 1980) ou *ranksum* de Wilcoxon (WILCOXON, 1945), para testes paramétricos e não paramétricos, respectivamente.

Infelizmente, não é possível extrair informações relevantes sobre o real desempenho dos algoritmos com tais métricas. É possível entender que o DSRBF tem capacidade de, em geral, predizer corretamente a classe de uma maior quantidade de instâncias. Entretanto, como se tratam de bases desbalanceadas, não há como saber quais dos algoritmos conseguem aprender sobre a maior quantidade de classes. Ou seja, a partir da acurácia, não há como saber se algum algoritmo favoreceu o aprendizado de somente algumas poucas classes.

A complementação da Sensibilidade Mínima fornece informação sobre o pior desempenho de alguma classe. Novamente, não é possível extrair informação mais relevante além desta. Pode acontecer de todas as menores classes serem aprendidas de forma insuficiente; porém, não há como saber disso, uma vez que somente o pior desempenho é levado em consideração pela métrica. Em contrapartida, o algoritmo concorrente pode ter sucesso no aprendizado de todas as classes, exceto uma ou duas, e, por isso, sua Sensibilidade Mínima ter um valor menor. Desta forma, o algoritmo que conseguiu um desempenho real melhor pode ficar prejudicado. Por fim, é importante destacar também a grande variabilidade da média da Sensibilidade Mínima, expressa através do desvio-padrão. Tal variabilidade

Tabela 22 – Resumo das bases de dados usadas pelos algoritmos AMDO, MDO e MDO+

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Balance Scale	3	4	625	5,8776	2,6985
Car Evaluation	4	6	1728	18,6154	10,3890
Contraceptive	3	9	1473	1,8889	0,2159
Dermatology	6	34	358	5,55	1,8418
E.coli	5	7	327	7,15	2,7030
Flare	6	11	1066	7,6977	3,3327
Gene	3	60	3190	2,1578	0,4134
Hayes Roth	3	4	132	1,7	0,1922
New Thyroid	3	5	215	5	1,9143
Nursery	4	8	12958	13,1707	8,1868
Thyroid	3	21	7200	40,1566	18,3396

impede a estimação do real desempenho.

Portanto, para esta comparação, a conclusão é que o DSRBF é capaz de uma boa classificação *geral*, porém não é possível distinguir o quão bons foram os resultados para bases desbalanceadas com múltiplas classes. Por isso, apesar de ser sempre melhor em acurácia que o DCIA, não é possível dizer que o algoritmo é, de fato, melhor para este problema.

Além do DSRBF, o DCIA ainda é comparado a outros três algoritmos em nível de dados encontrados na literatura: MDO (ABDI; HASHEMI, 2016), MDO+ e AMDO (YANG et al., 2018). Os dois últimos dos três algoritmos citados são uma modificação e evolução do MDO, respectivamente. Não foi possível encontrar o código fonte dos mesmos, e os resultados do MDO fornecidos em seu trabalho original são ranks das médias obtidas. Contudo, o trabalho no qual o algoritmo AMDO é apresentado fornece os desempenhos dos três algoritmos através da métrica MAUC.

Os experimentos foram conduzidos com *5-fold cross-validation* com dez repetições. No trabalho original os autores utilizaram 15 bases de dados. Entretanto, por limitações de tempo, somente onze foram utilizadas pelo DCIA. A Tabela 22 apresenta as bases utilizadas nesta comparação. É importante notar também que os autores do AMDO excluíram as classes que possuem menos de dez instâncias. Devido a isto, bases como *E.coli* e *Nursery* estão apresentadas com valores diferentes. Ainda, instâncias com *missing values* também foram excluídas, resultando em menos instâncias nas bases *Dermatology* e *Hayes-Roth*.

Os algoritmos MDO e MDO+ não são capazes de executar sobre bases de dados com atributos mistos, i.e., atributos numéricos e nominais. Por outro lado, o AMDO consegue lidar com ambos os tipos de atributos. Devido a isto, as comparações com MDO e MDO+ foram feitas de forma separada.

Tabela 23 – Desempenhos do MDO, MDO+, DCIACSO, DCIAPSO e DCIASGSA utilizando a média e desvio-padrão da métrica MAUC

Base	**MDO**	**MDO+**	**DCIACSO**	**DCIAPSO**	**DCIASGSA**
Balance Scale	0,6794±0,0211	0,6690±0,0139	0,8066±0,0429	0,7684±0,0427	**0,8300**±0,0218
Dermatology	**0,9759**±0,0124	**0,9772**±0,0096	**0,9701**±0,0241	0,9461±0,0430	0,9538±0,0306
E.coli	0,8795±0,0423	0,8941±0,0305	**0,9194**±0,0277	0,8710±0,0262	0,8763±0,0329
Hayes Roth	**0,8834**±0,0484	**0,8876**±0,0461	0,8353±0,0432	0,8413±0,0508	0,8022±0,0581
New Thyroid	0,9140±0,0467	**0,9616**±0,0262	**0,9576**±0,0225	0,9493±0,0309	0,9100±0,0584

A Tabela 23 apresenta as médias e desvios-padrões da métrica MAUC obtidos pelo DCIA e MDO. De forma similar às comparações com o DSRBF, os valores das médias e desvios-padrões do MDO e MDO+ foram extraídos do artigo (YANG et al., 2018). Com esses valores foram criadas distribuições normais. Devido a isto, é possível que os desempenhos apresentados não sejam idênticos aos do artigo original. Por fim, o DCIA foi executado novamente seguindo a nova configuração de experimento.

As comparações estatísticas foram realizadas com ANOVA quando todos os desempenhos tendiam para uma distribuição normal ou com Kruskal Wallis quando ao menos um dos desempenhos não tinha esta distribuição. Em **negrito** estão destacados os melhores desempenhos obtidos. O DCIA, com duas variações, obteve o melhor desempenho em quatro das cinco bases utilizadas. Das variações, a que utiliza CSO é a que mais se destaca.

Em relação ao MDO, o DCIA conseguiu se destacar nas bases com maior desbalanceamento, i.e., as que possuem maior valor de MIR. Entretanto, levando-se em conta todas as bases ao mesmo tempo, ou seja, observando-se os ranks, todos os algoritmos obtiveram desempenho estatisticamente similar, como pode ser visto na Figura 37.

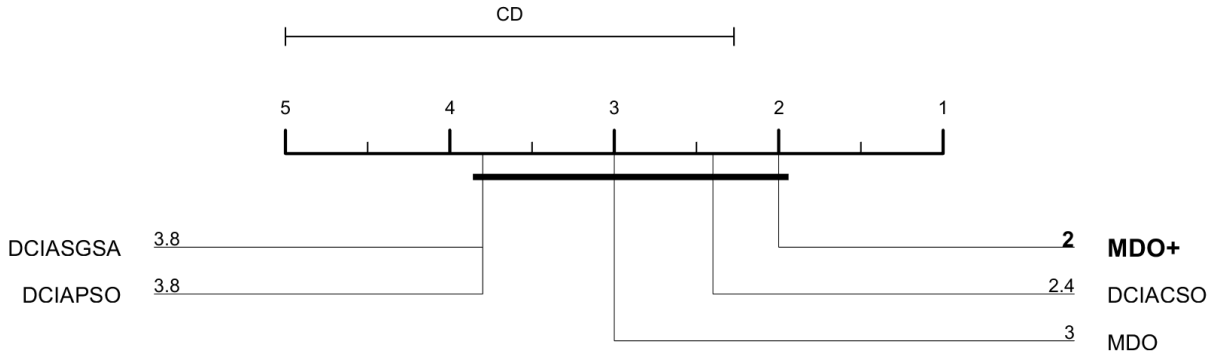

Figura 37 – CDD dos desempenhos obtidos pelo MDO, MDO+, DCIACSO, DCIAPSO e DCIASGSA.

Por fim, o DCIA é comparado ao AMDO (YANG et al., 2018), evolução do MDO. As configurações do experimento continuam as mesmas do MDO e MDO+, exceto pela maior quantidade de bases de dados utilizada. As comparações estatísticas, neste caso, foram

Tabela 24 – Desempenhos do AMDO, DCIACSO, DCIAPSO e DCIASGSA utilizando a média e desvio-padrão da métrica MAUC

Base	**AMDO**	**DCIACSO**	**DCIAPSO**	**DCIASGSA**
Balance Scale	0,7071±0,0194	**0,8066**±0,0429	**0,7684**±0,0427	**0,8300**±0,0218
Car Evaluation	0,9462±0,0072	**0,9549**±0,0160	0,9485±0,0241	0,9334±0,0165
Contraceptive	**0,6428**±0,0148	0,6356±0,0138	0,6221±0,0345	0,6287±0,0180
Dermatology	**0,9813**±0,0015	0,9701±0,0241	0,9461±0,0430	0,9538±0,0306
E.coli	0,8862±0,0290	**0,9194**±0,0277	0,8710±0,0262	0,8763±0,0329
Flare	0,7801±0,0098	**0,8959**±0,0083	**0,8910**±0,0084	**0,8987**±0,0082
Gene	**0,9615**±0,0050	0,7660±0,0261	0,7626±0,0213	0,7676±0,0369
Hayes Roth	**0,8849**±0,0515	0,8353±0,0432	0,8413±0,0508	0,8022±0,0581
New Thyroid	**0,9692**±0,0219	0,9576±0,0225	0,9493±0,0309	0,9100±0,0584
Nursery	**0,9714**±0,0082	0,8207±0,0101	0,8158±0,0203	0,8190±0,0054
Thyroid	0,9679±0,0067	**0,9720**±0,0090	0,9560±0,0159	0,9663±0,0132

feitas em pares, de forma similar às comparações com o DSRBF. A Tabela 24 apresenta os resultados obtidos por cada algoritmo.

Os resultados do DCIA destacados em **negrito** indicam superioridade estatística sobre o AMDO. Um resultado sublinhado indica equivalência estatística entre a versão do DCIA e o AMDO. Por fim, os resultados destacados do AMDO indicam quando o mesmo foi estatisticamente superior às três versões do DCIA.

Em cinco das onze bases o DCIA foi melhor, enquanto o AMDO obteve o melhor desempenho em seis bases. Entretanto, em cinco bases a vantagem de desempenho obtido por qualquer dos algoritmos é bem pequena. Por exemplo, na base *Car Evaluation* o DCIACSO obteve 0,955 de desempenho, enquanto o AMDO obteve 0,946. Outro exemplo são os desempenhos na base *Dermatology*, onde o AMDO obteve 0,98 enquanto o DCIACSO obteve 0,97. Ao se desconsiderar essas bases há um certo empate, onde cada algoritmo se destaca em três das onze bases.

Ao se observar as características das bases, é possível perceber que o DCIA, principalmente o DCIACSO, obtém um melhor desempenho para as bases com maior desbalanceamento. Mais especificamente, em todas as bases cujo valor de MIR é maior que 2, exceto *Nursery*, o DCIA obtem um melhor desempenho.

Por fim, levando-se em consideração todos os experimentos desta seção, é possível ressaltar os seguintes pontos:

- Existe dificuldade em se realizar comparações com algoritmos mais sofisticados encontrados na literatura. Esta dificuldade reside sobre como as informações do algoritmo e de seus resultados são publicados. Outro fator que prejudica a análise experimental é a falta de padronização das métricas utilizadas e configurações dos

experimentos;

- Parte da dificuldade citada impediu uma melhor comparação entre os desempenhos do DCIA e DSRBF;
- O DCIA, principalmente a variante com CSO, consegue um desempenho geral similar aos seus competidores diretos, ou seja, MDO, MDO+ e AMDO;
- Entretanto, o DCIA obtém os melhores desempenhos nas bases mais desbalanceadas.

4.3 TERCEIRA ABORDAGEM

Esta última seção de comparações considera as soluções *estado-da-arte* em nível de algoritmo. Tais soluções abordam o problema de múltiplas classes desbalanceadas, entretanto não são competidores diretos do DCIA, por causa de sua abordagem diferente.

Estes algoritmos adaptam classificadores para melhorarem seu poder de generalização em bases com múltiplas classes desbalanceadas. Ao mesmo tempo, levam em consideração que o conjunto de treinamento e teste não sofrerá qualquer tipo de modificação.

A contraposição do DCIA com os mesmos deve ser vista como o desempenho do 1NN, um dos mais simples classificadores, comparado ao desempenho de classificadores construídos especificamente para este problema. Caso o 1NN consiga ter um desempenho similar ou melhor, é possível inferir que o DCIA é bastante benéfico como algoritmo de pré-processamento, uma vez que permite a um dos mais simples classificadores ter um desempenho similar a outros classificadores mais complexos.

As comparações são realizadas entre o DCIA e AdaBoost.NC (WANG; YAO, 2012), AdaC2.M1 (SUN; KAMEL; WANG, 2006), AMCS (YIJING et al., 2016), CoMBo (KOÇO; CAPPONI, 2013) e DECOC (BI; ZHANG, 2018). Estes algoritmos foram selecionados por apresentarem bons desempenhos no tratamento do problema abordado, tinham seu código-fonte ou seus resultados publicados com médias e desvios-padrões dos resultados, o que possibilita a simulação dos desempenhos. Todos os algoritmos citados constroem comitês de classificadores para tratar o problema investigado. Devido a isto, não é esperado que o 1NN com o DCIA consiga superá-los em desempenho, principalmente os mais novos ou mais complexos, como AMCS e DECOC. Entretanto, é interessante verificar como os desempenhos se comparam.

A primeira comparação é feita entre as versões do DCIA e o algoritmo AdaBoost.NC. Os experimentos foram realizados em vinte bases de dados, apresentadas na Tabela 25. Os dados foram divididos com a técnica *5-fold cross-validation*. O código-fonte do AdaBoost.NC pode ser encontrado em <*https://github.com/chongshengzhang/Multi_Imbalance*>.

As comparações foram realizadas em pares, com os testes *Student's t-test* e *Wilcoxon's ranksum* para dados com distribuições normais e não normais, respectivamente. Os resultados comparados são apresentados nas Figuras 38 a 40. Para cada base de dados a

Tabela 25 – Resumo das bases de dados usadas pelos algoritmos AdaBoost.NC e AdaC2.M1

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Abalone	23	8	4172	344,5	268,5644
Balance Scale	3	4	625	5,8776	2,6985
Car Evaluation	4	6	1728	18,6154	10,3890
Contraceptive	3	9	1473	1,8889	0,2159
Dermatology	6	34	366	5,6	1,8598
E.coli	8	7	336	71,5	47,3469
Gene	3	60	3190	2,1578	0,4134
Glass	6	9	214	8,4444	5,0133
Hayes-Roth	3	4	160	2,0968	0,3743
Horse	3	14	366	4,3269	1,2592
Nursery	4	8	12958	13,1707	8,1868
Page Blocks	5	10	5473	175,4643	59,5996
Post Operative	3	8	90	32	13,7188
Satimage	6	36	6435	2,4489	0,9564
Shuttle	7	9	58000	4558,6	1676,8
Soybean	15	35	630	4,6	7,6660
Thyroid	3	21	7200	40,1566	18,3396
Wine	3	13	178	1,4792	0,0774
Yeast	10	8	1484	92,6	44,7543
Zoo	7	16	101	10,25	4,9225

média obtida por cada algoritmo é marcada, e seus desvios-padrões são indicados com as hastes.

Os algoritmos foram estatisticamente equivalentes em doze das bases de dados. Das oito base restantes, o DCIA foi estatisticamente melhor em quatro bases (*Balance Scale*, *Contraceptive*, *Page Blocks* e *Yeast*) enquanto o AdaBoost.NC obteve melhor desempenho em outras quatro bases (*Gene*, *Hayes Roth*, *Nursery*, *Satimage*). Alguns padrões podem ser notados quando observadas as bases em que os algoritmos se destacaram. O primeiro padrão consiste em o DCIA superar o AdaBoost.NC nas bases mais desbalanceadas, exceto na base *Contraceptive*.

Outro padrão que pôde ser observado consiste no AdaBoost.NC conseguir um melhor desempenho quando os atributos das instâncias estão dispostos em valores discretos. Por fim, quando os valores dos atributos possuem uma variação considerável, a tendência é de o DCIA obter um desempenho melhor. A exceção se deu na base *Satimage*, onde o AdaBoost.NC obteve melhor desempenho, entretanto sem muita vantagem, pois a diferença entre as médias é de pouco mais de 0,01.

A próxima comparação realizada é feita entre as variantes do DCIA e o algoritmo

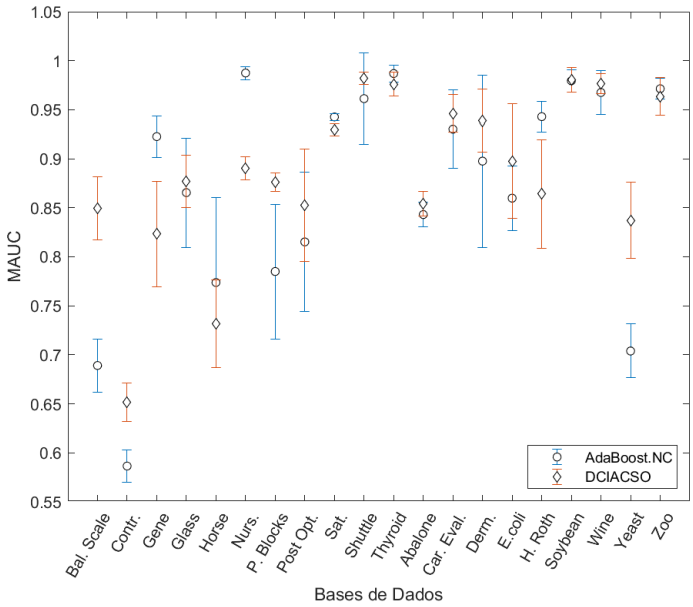

Figura 38 – Comparação entre os desempenhos obtidos pelo AdaBoost.NC e DCIACSO.

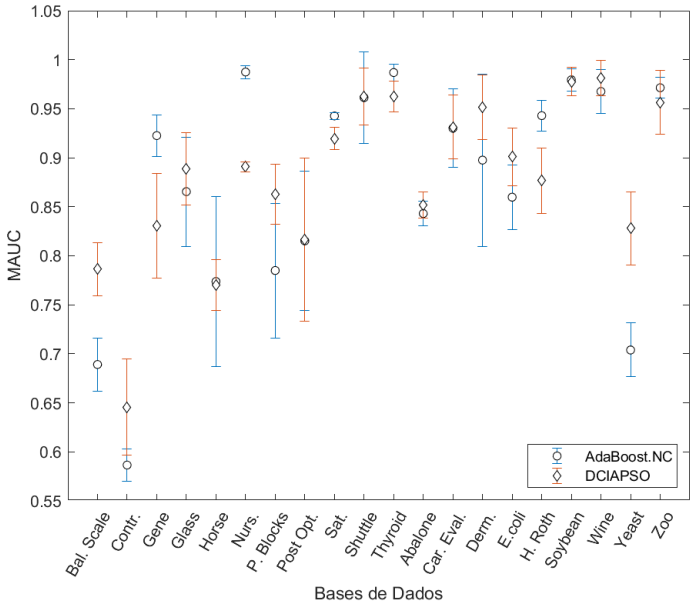

Figura 39 – Comparação entre os desempenhos obtidos pelo AdaBoost.NC e DCIAPSO.

AdaC2.M1. O código-fonte deste algoritmo pode também ser encontrado no endereço <*https://github.com/chongshengzhang/Multi_Imbalance*>. As bases de dados utilizadas são as mesmas da comparação com o AdaBoost.NC, e estão apresentadas na Tabela 25. Os dados foram divididos com a técnica *5-fold cross-validation*. As comparações estatísticas seguiram o mesmo modelo, e os resultados podem ser visualizados nas Figuras 41 a 43. O que se pode perceber é que as comparações são bastante similares às anteriores, com o AdaBoost.NC. As observações, portanto, são as mesmas.

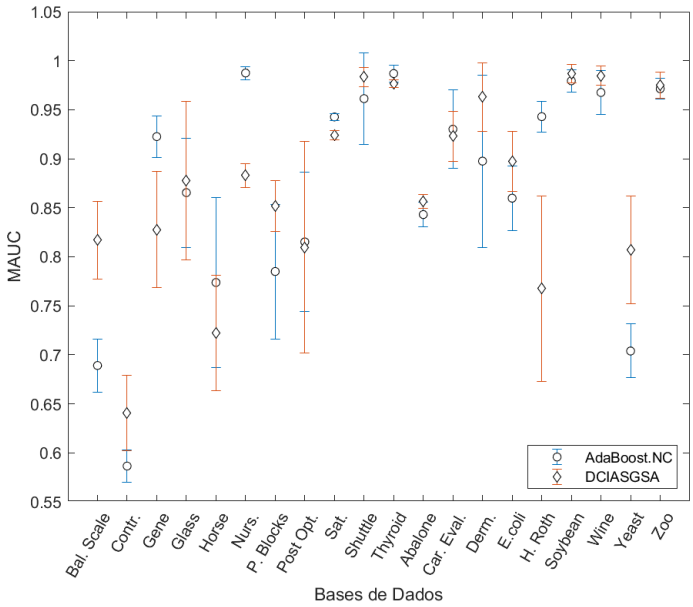

Figura 40 – Comparação entre os desempenhos obtidos pelo AdaBoost.NC e DCIASGSA.

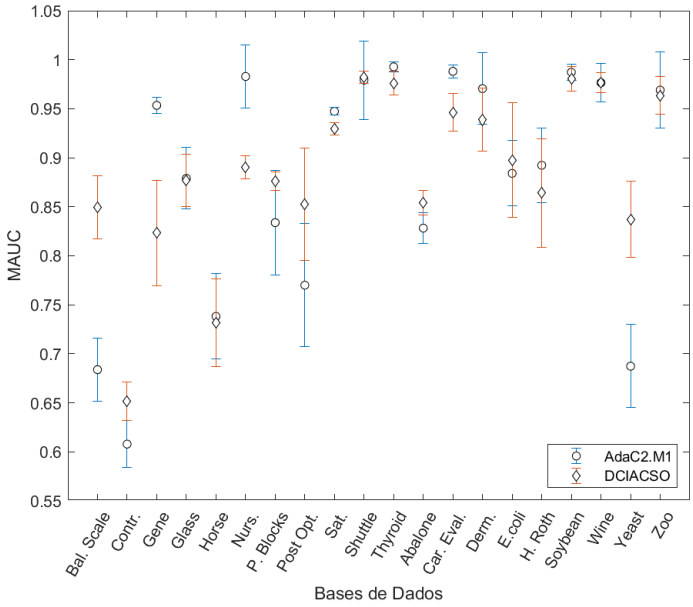

Figura 41 – Comparação entre os desempenhos obtidos pelo AdaC2.M1 e DCIACSO.

Os autores do AMCS disponibilizaram o código-fonte do seu algoritmo no endereço <*https://github.com/liyijing024/AMCS*>. Foram disponibilizadas também as bases pré-processadas que foram utilizadas, as quais são apresentadas na Tabela 26. Note que na referida tabela foram dispostas apenas as bases que ainda não foram apresentadas em qualquer outra tabela deste capítulo ou as bases que sofreram algum tipo de modificação. Ao todo foram utilizadas vinte e duas bases nas comparações.

As bases excluídas da Tabela 26 mas cujas informações já estão presentes em outras

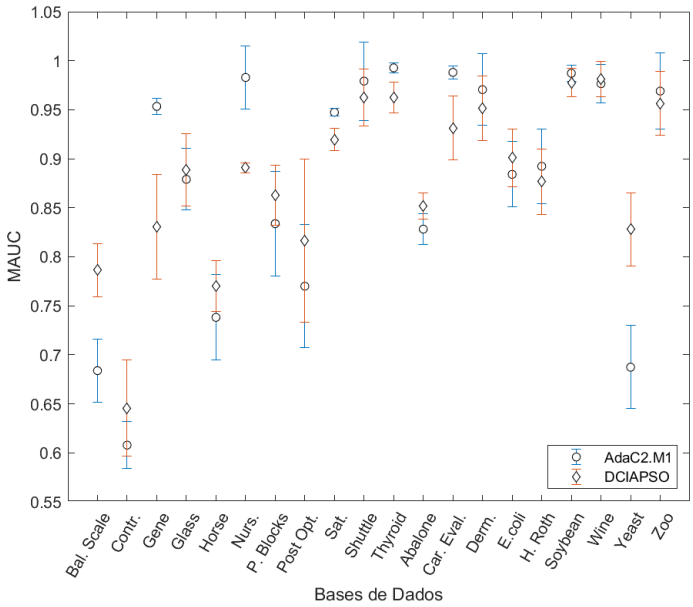

Figura 42 – Comparação entre os desempenhos obtidos pelo AdaC2.M1 e DCIAPSO.

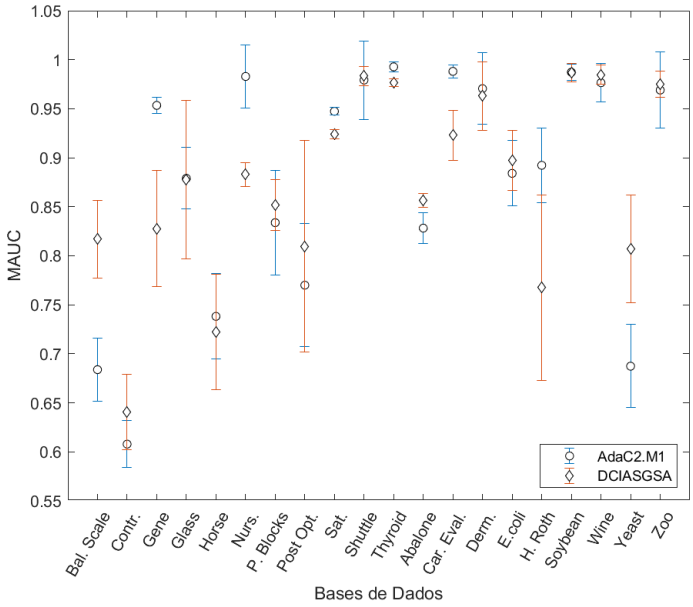

Figura 43 – Comparação entre os desempenhos obtidos pelo AdaC2.M1 e DCIASGSA.

tabelas são: *Autos*, *Balance Scale*, *Contraceptive*, *Landsat*, *Lymphography*, *Page Blocks*, *Penbased*, *Wine*, *Wine QR*, *Yeast* e *Zoo*. Os dados foram divididos com *5-folds* SCV, e as comparações foram feitas em pares. Os testes de hipóteses utilizados foram *Student's t-test* e *Wilcoxon's ranksum* para resultados com distribuição normal e não normal, respectivamente. Desta vez, os resultados estão expressos através da métrica AUCarea, como no artigo do experimento original.

As comparações estão apresentadas na Figura 44. Nesta figura a linha diagonal re-

Tabela 26 – Resumo de algumas das bases de dados usadas pelo algoritmo AMCS

Base	**#Classes**	**#Atributos**	**#Instâncias**	**IR**	**MIR**
Abalone	28	8	4182	344,5	560,6202
Car Evaluation	4	6	1152	29,6	15,3789
Dermatology	6	34	358	5,55	1,8418
E.coli	8	7	335	71	47,1842
Gene	4	60	1000	1,25	0,0317
Hayes Roth	3	5	132	1,7	0,1922
MC-FAC	3	216	2000	8	4,0833
MF-KAR	3	64	2000	8	4,0833
New Thyroid	3	4	215	5	1,9143
Soybean	19	35	308	20	16,3930
Wine QW	5	11	4718	109,9	50,1182

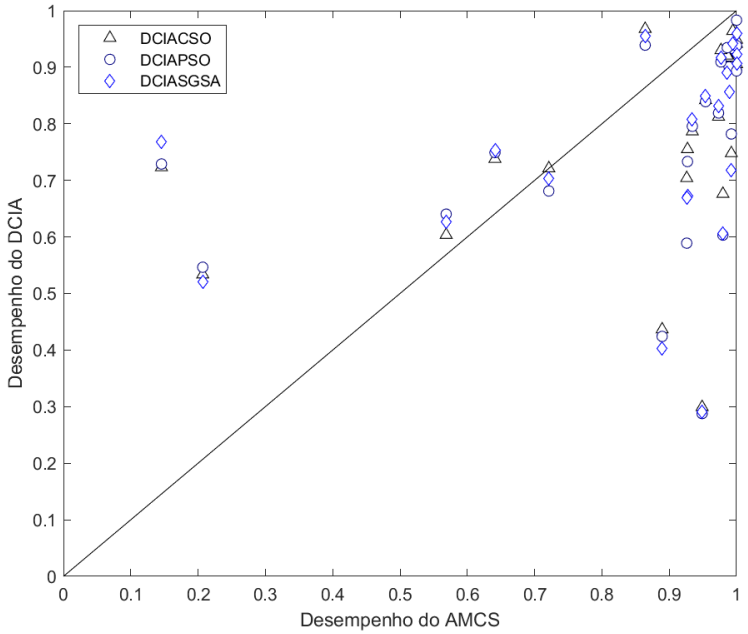

Figura 44 – Comparação entre os desempenhos obtidos pelo AMCS, DCIACSO, DCIAPSO e DCIASGSA.

presenta o desempenho do AMCS, enquanto as marcações representam os desempenhos obtidos pelo DCIA, em cada base de dados. Sempre que a média de desempenho do DCIA for maior que a do AMCS, sua marcação estará acima da linha. Caso a média de desempenho seja similar, a marcação está sobre a linha. Por fim, caso o desempenho tenha sido pior, a marcação ficará sob a linha diagonal.

O DCIACSO superou estatisticamente o AMCS nas seguintes bases: *Abalone*, *Wine QW* e *Page Blocks*. Além dessas, os algoritmos foram estatisticamente similares em *New*

Thyroid, *Wine QR*, *Yeast* e *Zoo*.

Quanto ao DCIAPSO, o AMCS foi superado estatisticamente nas bases *Abalone*, *Wine QW* e *Page Blocks*. Os desemepenhos foram estatisticamente similares nas bases *New Thyroid*, *Wine*, *Wine QR*, *Yeast* e *Zoo*.

Por fim, o DCIASGSA superou o AMCS nas bases *Abalone*, *Wine QW*, e *Page Blocks*. Ao mesmo tempo seus desempenhos foram estatisticamente similares nas bases *Dermatology*, *WineQR*, *Yeast* e *Zoo*.

Uma tendência que também pôde ser observada nestes experimentos é que quanto mais desbalanceadas são as classes, melhor é o desempenho do DCIA em relação aos seus competidores. Existem algumas exceções, onde o DCIA consegue o melhor desempenho em bases pouco desbalanceadas, ou o algoritmo em comparação tendo o melhor desempenho em bases bastante desbalanceadas. Contudo, o melhor desempenho obtido na maior parte das classes mais desbalanceadas pertence a alguma variação do DCIA.

Por exemplo, o AMCS obteve o melhor desempenho em dezenove bases de dados. Dessas, 63% possuem um valor de MIR menor ou igual a 10. Ao mesmo tempo, das bases em que o DCIACSO obteve o melhor desempenho, 71% possuem o valor de MIR acima de 41. A mesma observação é válida para o DCIASGSA, e há uma pequena diferença para o DCIAPSO, pois o mesmo é estatisticamente equivalente ao AMCS na base *Wine*, cujo valor de MIR é 0,0774.

Entretanto, o DCIA não deve ser entendido como um algoritmo ruim para bases pouco desbalanceadas. Apesar de o mesmo não atingir o mesmo desempenho que seus competidores em várias bases cujo valor de MIR é baixo, em várias situações o desempenho obtido não é tão inferior. É possível observar na Figura 44, no canto direito superior, que o DCIA consegue desempenhos próximos ao do AMCS, apesar de inferiores.

Existem ainda quatro casos especiais. Nas bases *Wine QW* e *Page Blocks* o AMCS teve um péssimo desempenho, com uma média abaixo de 0,21 em termos da métrica AUCarea. Ambas as bases possuem muitas semelhanças, como a mesma quantidade de classes, uma quantidade quase idêntica de atributos, um grande número de instâncias e um alto valor de MIR. Os atributos são mistos, ou seja, discretos e contínuos. Os seus valores possuem uma enorme variação. Neste caso, a normalização que ocorre durante a execução do DCIA possivelmente contribuiu para que o 1NN tivesse uma melhor generalização.

Os outros dois casos especiais se dão nas bases *Gene* e *Contraceptive*, nas quais todas as variações do DCIA obtiveram um péssimo desempenho. É importante lembrar, entretanto, que a métrica AUCarea é pessimista, ou mais sensível, em relação à métrica MAUC, o que explica valores tão baixos. Contudo, ao se observar as características dessas bases, é possível perceber dois principais fatores que podem contribuir para o baixo desempenho: todos os atributos são discretos e o valor do MIR é bastante baixo, abaixo de 1. Decerto, a disposição das classes no espaço amostral, como também a disposição de suas bordas podem ter alguma influência. Isto explica o bom desempenho do DCIA em outras bases

Tabela 27 – Desempenhos do CoMBo, DCIACSO, DCIAPSO e DCIASGSA através da média e desvio-padrão da métrica MAUC

Base	**CoMBo**	**DCIACSO**	**DCIAPSO**	**DCIASGSA**
Balance Scale	**0,8847**±0,0328	0,8244±0,0402	0,7823±0,0432	0,8169±0,0462
Car Evaluation	**0,9932**±0,0023	0,9470±0,0270	0,9380±0,0250	0,9267±0,0331
E.coli	**0,9635**±0,0138	0,9150±0,0335	0,9130±0,0346	0,9163±0,0319
Glass	**0,9472**±0,0248	0,8748±0,0394	0,8726±0,0343	0,8689±0,0443
Satimage	**0,9757**±0,0027	0,9276±0,0076	0,9239±0,0096	0,9235±0,0109
New Thyroid	**0,9958**±0,0096	0,9639±0,0316	0,9571±0,0435	0,9592±0,0380
Nursery	**1,0000**±0,0000	0,8281±0,0154	0,8098±0,0278	0,8256±0,0137
Yeast	**0,8616**±0,0254	0,8253±0,0350	0,8155±0,0312	0,8114±0,0359

que possuem características similares, como a base *Dermatology*. Um estudo aprofundado sobre todas as características das bases é necessário para se entender com mais precisão sua influência sobre os desempenhos obtidos.

Após o AMCS, o DCIA é comparado ao CoMBo (KOÇO; CAPPONI, 2013). Nenhum código-fonte foi disponibilizado até então, contudo no artigo original os desempenhos foram apresentados com médias e desvios-padrões, o que possibilita uma simulação de seu desempenho. Os dados foram separados em *5-folds* e o experimento repetido dez vezes. As comparações com o DCIA foram feitas em oito bases de dados, as quais foram apresentadas na Tabela 18. Entretanto as bases *E.coli* e *Nursery* foram modificadas, de forma que as classes raras foram excluídas. A partir disso a base *E.coli* passou de oito para cinco classes, enquanto a base *Nursery* passou de cinco para quatro classes.

As comparações foram feitas em pares, e os testes de hipóteses utilizados foram *Student's t-test* e *Wilcoxon's ranksum*, para resultados com distribuição normal e não normal, respecitvamente. A Tabela 27 apresenta os resultados obtidos. Desta vez, nenhuma das versões do DCIA obteve desempenho similar ou melhor. Devido a isto, todos os desempenhos do CoMBo foram destacados, tornando este algoritmo o único que supera o DCIA em todas as comparações.

A média das diferenças de desempenhos, desconsiderando a base *Nursery*, é 0,0553. Ou seja, na métrica MAUC as versões do DCIA obtém uma média de desempenho relativamente próxima à do CoMBo, ainda que em nenhum dos casos seja estatisticamente equivalente ou superior. Somente em uma das bases houve uma grande diferença entre os desempenhos. Entretanto, levando-se em conta se tratar de uma base cujos atributos são discretos, pode-se dizer que o DCIA manteve uma coerência em seu desempenho. É possível perceber esta coerência ao se observar o desempenho do DCIA em outras bases cujos atributos também são discretos, como *Balance Scale*. Outra vez, esta característica dos atributos não deve ser vista como a única razão para tal desempenho.

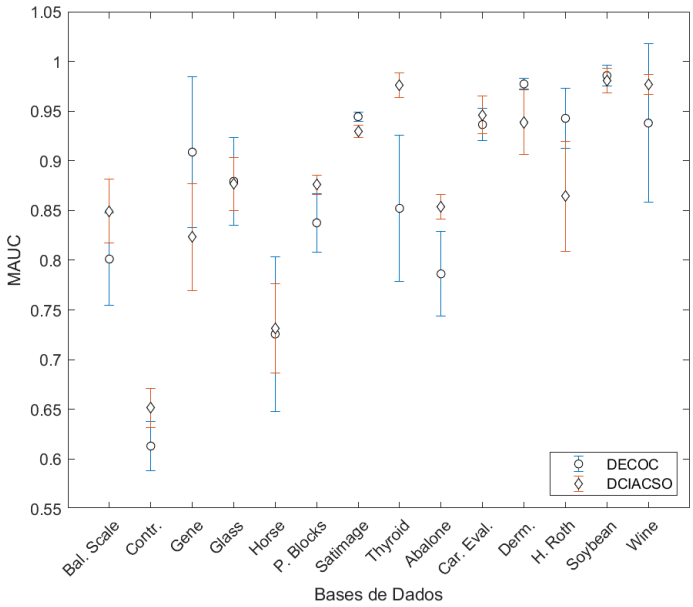

Figura 45 – Comparação entre os desempenhos obtidos pelo DECOC e DCIACSO.

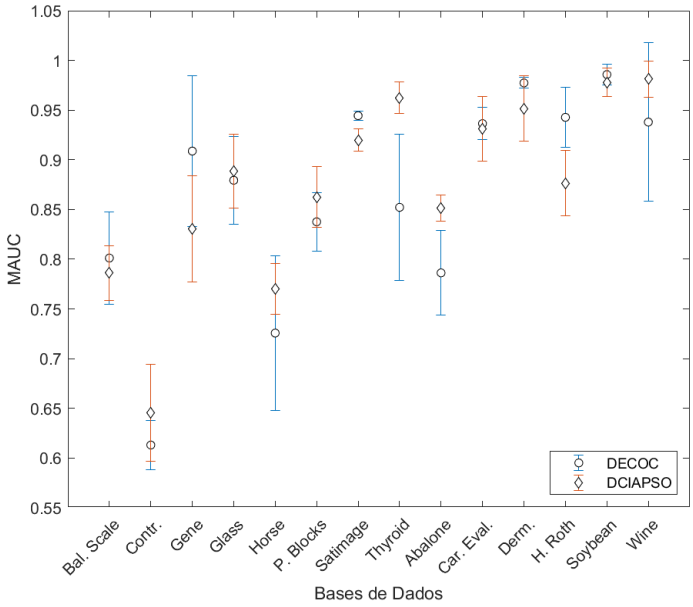

Figura 46 – Comparação entre os desempenhos obtidos pelo DECOC e DCIAPSO.

O último algoritmo a ser comparado é o DECOC (BI; ZHANG, 2018). O seu código-fonte está disponível em <*https://github.com/chongshengzhang/Multi_Imbalance*>. Além do DECOC, os autores também disponibilizaram sua implementação de outros algoritmos, como o AdaBoost.NC e AdaC2.M1, os quais foram utilizados nesta pesquisa. Os autores justificam a disponibilização como uma medida para o avanço da pesquisa neste campo de estudo (BI; ZHANG, 2018). É importante mencionar que os códigos-fonte do VDBC e o DCIA também estão disponíveis na internet.

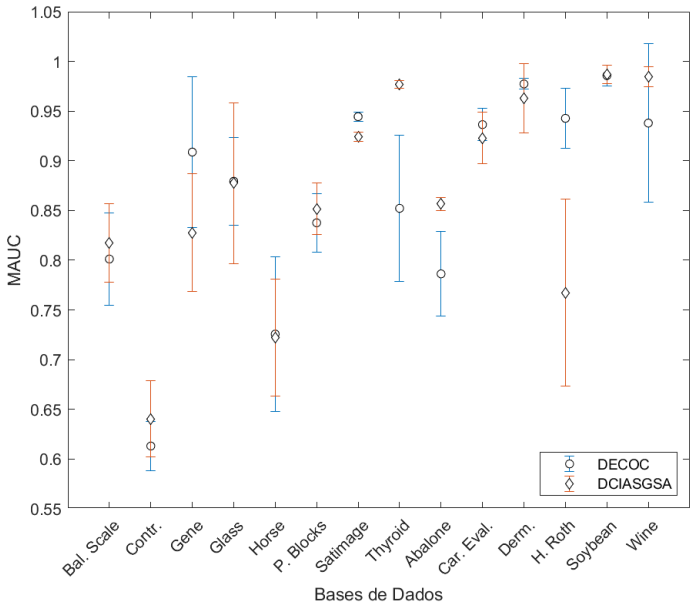

Figura 47 – Comparação entre os desempenhos obtidos pelo DECOC e DCIASGSA.

Os algoritmos DECOC e as variações do DCIA foram executados em 14 bases de dados, as quais podem ser consultadas na Tabela 18. Os dados foram divididos com a técnica *5-folds* SCV. Os testes de hipótese foram feitos com *Student's t-test* e *Wilcoxon's ranksum*, para resultados com distribuição normal e não normal, respectivamente. As comparações podem ser visualizadas nas Figuras 45 a 47.

Os testes estatísticos mostram que o DCIACSO obtém desempenho melhor nas bases *Abalone*, *Contraceptive* e *Page Blocks*. Por outro lado, o DECOC obteve desempenho estatisticamente superior nas bases *Dermatology*, *Hayes-Roth* e *Satimage*. Nas demais bases ambos obtiveram desempenhos estatisticamente equivalentes.

Apesar de na base *Thyroid* os mesmos serem estatisticamente equivalentes, na Figura 45 é possível perceber claramente que o DCIACSO obtém um melhor desempenho. O teste de hipóteses acusou equivalência devido a um *outlier* do DECOC. O *p-value* do teste é pouco acima de 0,05, ou seja, pouco acima do limite para que se considere duas amostras como pertencentes a diferentes populações.

O DCIAPSO obteve um melhor desempenho estatisticamente somente na base *Abalone*. O DECOC, por sua vez, foi melhor nas bases *Hayes-Roth* e *Satimage*. Nas demais bases ambos foram estatisticamente equivalentes. O DCIASGSA obteve os mesmos resultados nas comparações estatísticas. Ainda, o mesmo caso do DCIACSO sobre a base *Thyroid* acontece de forma similar com essas duas versões do DCIA.

Ao se observar as bases nas quais alguma versão do DCIA obteve melhor desempenho, as seguintes características gerais puderam ser constatadas. O desbalanceamento comumente é alto, a partir da medição feita com o MIR, e os atributos são contínuos ou mistos. No único caso em que o DCIA obteve um melhor desempenho em uma base com um va-

lor de MIR baixo, foram observados dois atributos cujos valores possuem uma variação considerável.

Também é interessante notar as comparações na base *Wine*. O alto valor de desvio-padrão no desempenho do DECOC tem contido a média e desvio-padrão, para mais e para menos, de todas as variações do DCIA. Isto sugere que, nesta base, o DCIA também pode ser considerado como tendo um desempenho melhor.

As comparações com o DECOC marcam o fim dos experimentos da Terceira Abordagem. A partir do que foi exposto, é possível destacar os seguintes pontos:

- Quanto maior o desbalanceamento entre as classes de uma base, calculado com o MIR, maior é a tendência do DCIA superar estatisticamente os desempenhos das soluções em nível de algoritmo;
- O DCIA aparentemente não é muito eficaz em bases que possuem atributos cujos valores sejam discretos. Entretanto, quando tais valores possuem uma variação considerável o DCIA consegue alguma vantagem, possivelmente por ter a normalização dos dados como um de seus passos;
- Esta Terceira Abordagem pode ser vista como uma comparação entre o 1NN e algoritmos que criam comitês de classificadores. Além de utilizarem vários classificadores, tais algoritmos são adaptados para o problema de múltiplas classes desbalanceadas. Logo, a comparação se deu entre um dos mais simples classificadores e um conjunto de classificadores mais complexos e adaptados para o problema. Percebe-se, portanto, que o pré-processamento do conjunto de treinamento realizado pelo DCIA possibilita ao 1NN alcançá-los em desempenho, e algumas vezes, até mesmo superá-los;
- É possível notar que as características das bases possuem certa influência padronizada no desempenho obtido pelos classificadores. Portanto, pode ser interessante que, em algum trabalho futuro, o DCIA seja também adaptado às bases como é o método AMCS.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo as três melhores versões do DCIA foram comparadas a vários algoritmos encontrados na literatura. As comparações foram divididas em três abordagens, de forma a serem observados diferentes aspectos das comparações.

A primeira abordagem teve seu foco sobre a comparação do desempenho do DCIA em relação a outras soluções em nível de dados, porém simples e amplamente conhecidos. Em linhas gerais, foi possível observar que a proposta desta tese supera todos os seus concorrentes, levando-se em conta o classificador 1NN.

A segunda abordagem continua as comparações com soluções em nível de dados, entretanto, tendo como concorrente algoritmos mais complexos, encontrados na literatura. As dificuldades relatadas no início do capítulo são bastante aplicáveis neste momento, cujas comparações e conclusões foram prejudicadas.

A terceira abordagem comparou os desempenhos obtidos pelas versões do DCIA ao de soluções em nível de algoritmo. Desta vez foi possível observar o efeito do DCIA no desempenho do classificador 1NN, um dos mais simples conhecidos, em relação ao desempenho de classidicadores *ad hoc* bastante complexos. De forma geral, o DCIA mostrou-se bastante efetivo em permitir ao 1NN classificações, várias vezes, superior ao de modelos que se utilizam até mesmo de comitês de classificadores.

Em cada seção, que aborda mais detalhadamente cada abordagem, as principais conclusões são delineadas, com o fornecimento de mais informações, tanto sobre os experimentos, quanto sobre as conclusões. Após o fim de todos os experimentos conduzidos, desde aqueles durante a evolução do DCIA (Capítulo 3) àqueles de comparação com demais algoritmos existentes (este capítulo), é possível reunir todas as principais informações para uma subsequente conclusão.

5 CONSIDERAÇÕES FINAIS

O problema de classes desbalanceadas vem recebendo bastante atenção na literatura. Estudos com bases binárias (com duas classes) são recorrentes, enquanto bases com múltiplas classes ainda recebem menos atenção. As soluções encontradas na literatura podem ser divididas em duas abordagens, a saber, abordagem em nível de dados e abordagem em nível da algoritmo. A primeira se concentra em manipular a quantidade de instâncias das classes, enquanto a segunda foca em adaptar classificadores para lidar com o desbalanceamento.

Esta tese apresentou uma nova abordagem em nível de dados para o problema de múltiplas classes desbalanceadas. As abordagens em nível de dados normalmente focam em aumentar as menores classes, diminuir a quantidade de instâncias das maiores ou hibridizar usando ambas para redimensionar todas as classes de uma base (LÓPEZ et al., 2013). O objetivo desse tipo de abordagem é transformar um conjunto de treinamento de modo que as classes fiquem menos desbalanceadas. Desta forma, os problemas inerentes ao treinamento de modelos de Aprendizado de Máquina em bases de dados desbalanceadas são atenuados ou eliminados.

Outra maneira de se transformar um conjunto de treinamento, i.e., além de sobreamostragem e subamostragem, é através de métodos de redução de instâncias. Tais métodos podem ser divididos em Seleção de Protótipos (SP) e Geração de Protótipos (GP) (LÓPEZ et al., 2014). No contexto de bases de dados desbalanceadas, a redução de instâncias pode ser vista como uma terceira via da abordagem em nível de dados.

A utilização de SP ou GP em modelos de classificação em bases desbalanceadas pode ser encontrada na literatura (MILLÁN-GIRALDO; GARCÍA; SÁNCHEZ, 2012; VERBIEST et al., 2012; OLIVEIRA et al., 2012; LÓPEZ et al., 2014; OLIVEIRA et al., 2015). Contudo, a maioria dos trabalhos foca em bases binárias. A partir da pesquisa realizada na literatura, na época de início do desenvolvimento da tese, não foram encontrados trabalhos que utilizem SP ou GP em bases com múltiplas classes desbalanceadas. Desta forma, o VDBC (SILVA; ZANCHETTIN, 2016b), possivelmente, foi o primeiro trabalho neste sentido, seguido do DCIA.

A nova abordagem em nível de dados foi apresentada em duas grandes fases, o VDBC e o DCIA. O VDBC consiste no primeiro método proposto. A partir dele foram testadas algumas hipóteses, com as quais foi possível desenvolver um algoritmo experimentalmente melhor a cada alteração realizada, chamado DCIA. Uma vez implementada a primeira versão do DCIA, seguiu-se uma série de análises, culminando nas versões DCIACSO, DCIAPSO.M10 e DCIASGSA.M5.

O conceito geral para o desenvolvimento da nova abordagem é transformar, da maneira mais simples possível, o conjunto de treinamento em um conjunto de protótipos que

promova uma melhora do desempenho de classificação do modelo utilizado. Nesta tese, por focar no pré-processamento dos dados, nos experimentos de classificação foi utilizado um dos modelos de classificação mais simples, o 1NN, como forma de enfatizar a qualidade da proposta de representação dos dados baseados em reamostragem. Para isso, tanto o VDBC quanto DCIA, e todas as suas variações, foram examinadas através de testes estatísticos. Neste trabalho, a análise teve seu foco sobre o desempenho dos algoritmos considerando a métrica MAUC (HAND; TILL, 2001). Após sua validação, suas melhores variações, i.e., DCIACSO, DCIAPSO.M10 e DCIASGSA.M5 foram comparados a vários algoritmos encontrados na literatura, os quais também abordam o problema de múltiplas classes desbalanceadas.

Conclusões detalhadas sobre os experimentos são relatadas ao fim dos capítulos de proposta e experimentos. Entretanto, alguns pontos merecem ser ressaltados. Por exemplo, apesar de ser um algoritmo computacionalmente custoso, o DCIA tem o potencial de atingir uma grande taxa de redução de instâncias. No caso mais extremo, mais de dez mil instâncias passam a ser representadas por uma média de apenas onze protótipos. Com poucos protótipos gerados o DCIA permite ao 1NN a obtenção de desempenhos, algumas vezes, superiores ao de comitês de classificadores modificados para melhor generalizarem diante de bases com classes desbalanceadas. Em relação a outros algoritmos em nível de dados, o DCIA se mostrou equivalente ou superior, levando-se em conta o desempenho obtido pelo classificador base.

Outra característica interessante, e possivelmente pioneira, é a utilização de mais de uma métrica na execução do algoritmo. Durante a Seleção de Atributos a métrica AUCarea foi capaz de influenciar no desempenho final. Contudo, esta mesma métrica pode influenciar negativamente no desempenho final quando utilizada na geração de protótipos.

O DCIA, principalmente em conjunto com o CSO, mostrou ser uma alternativa válida em relação ao algoritmos *estado-da-arte*, principalmente em bases de dados com um alto grau de desbalanceamento. Como uma terceira abordagem em nível de dados, o DCIA mostra que a utilização de técnicas de redução de instâncias é um campo promissor a ser pesquisado na área de múltiplas classes desbalanceadas.

Outro ponto que merece destaque é a questão da simplicidade da solução. Como pode ser observado desde a criação da hipótese (Seção 3.1), um dos objetivos do estudo é encontrar a solução mais simples possível. A quantidade de passos, ou algoritmos utilizados durante todo o processo do DCIA pode dar a impressão de que tal objetivo não foi cumprido. Entretanto todo o processo, como também os algoritmos, são todos bastante simples. A divisão dos dados e sua subsequente normalização são realizadas com algoritmos bastante simples, amplamente conhecidos e com abundante material disponível para consulta. A seleção de atributos acrescenta apenas mais custo computacional, entretanto o algoritmo utilizado, i.e., o CSO é bastante simples de ser implementado. Da mesma forma todo o processo de inicialização, ajuste e inserção de novos algoritmos também é

conduzida de maneira bastante simples. Além disso, os principais algoritmos de busca utilizados, além do CSO, ou seja, PSO e SGSA, são também bastante simples de serem implementados. Além das implementações citadas, resta somente o cálculo das métricas, as quais também são fáceis de serem codificadas, e possuem farto material disponível para consulta.

Por fim, é interessante pontuar novamente as principais contribuições desta tese, as quais já estão citadas, dentre outras contribuições, no primeiro capítulo[1]. A princípio, os estudos começaram com uma visão geral sobre o problema, ainda em bases binárias. A partir desse primeiro estudo pôde-se sugerir que os problemas relatados com o desbalanceamento podem ser observados a partir de uma taxa IR igual ou superior a 4. Um estudo similar em bases com múltiplas classes ainda necessita ser conduzido.

A taxa de desbalanceamento para múltiplas classes desbalanceadas, o MIR, também pode ser pontuada como uma das principais contribuições. Esta taxa pode ser vista ou analisada semelhantemente ao IR, ao mesmo tempo em que apresenta o quanto o tamanho das classes de uma base são homogêneos — quanto menor o valor, mais homogêneo e, portanto, mais balanceado. Outra contribuição é a sugestão da utilização do método *hold-out* 33%, quando nem todas as classes puderem ser apropriadamente representadas após a separação dos dados com o método *10-folds*.

Quanto ao tamanho das classes, na literatura aquelas com quantidades de instâncias bastante menores em relação às demais são vistas como classes raras. Entretanto, tais classes raras podem ter ainda uma grande quantidade de instâncias, desde que sejam muito menores que as demais. Ao mesmo tempo existem as classes que, de fato, possuem quantidades praticamente irrisórias de instâncias. Uma das contribuições deste estudo, mais especificamente uma contribuição topológica, é rotular classes que possuem menos de cinco instâncias como classes extremamente raras.

A última contribuição principal, que também merece novo destaque, é a criação do algoritmo de busca SGSA. Este algoritmo é uma simplificação do GSA, e produz resultados similares, ou seja, cujas comparações são estatisticamente equivalentes.

5.1 TRABALHOS FUTUROS

Alguns pontos da pesquisa e do desenvolvimento do método proposto ainda ficaram em aberto. A seguir, tais pontos são listados como propostas para trabalhos futuros, os quais poderão agregar bastante em novos conhecimentos, não somente no campo específico do DCIA, mas na área de pesquisa de problemas com múltiplas classes desbalanceadas.

- Apesar dos avanços ainda não existe um consenso sobre quando considerar uma base de dados como desbalanceada. A classificação e a mensuração do desempe-

[1] Nem todas as principais contribuições são pontuadas devido ao fato de já terem sido ressaltadas nos parágrafos anteriores.

nho de classificadores podem sofrer grandes modificações quando uma determinada base de dados passa a ser encarada como desbalanceada. Entretanto, em bases de dados cuja taxa de desbalanceamento é praticamente irrisória, tais modificações possivelmente apenas acrescentam esforços ou custos desnecessários. Em (SILVA; ZANCHETTIN, 2015) é sugerido que, em bases binárias, os classificadores mais sensíveis ao desbalanceamento têm uma maior probabilidade de experimentar os efeitos do desbalanceamento quando a classe minoritária possui cerca de 25% ou menos da quantidade de instâncias da classe majoritária. Em geral, tal limite é dependente do classificador. Contudo há casos em que mesmo havendo pouco desbalanceamento, a classe majoritária já *domina* o classificador. Um estudo mais aprofundado sobre o tema pode se tornar de grande valia para a comunidade científica, não somente sobre bases binárias, mas também sobre bases com três ou mais classes. A partir de tais estudos pode haver a possibilidade de uma automação do *modo* de aprendizado escolhido, de acordo com a base de dados;

- O 1NN, através do DCIA, obteve melhores desempenhos que seus competidores à medida em que a taxa de desbalanceamento se tornava maior. Entretanto, para bases de dados cujo valor do IR ou do MIR são baixos, o 1NN obteve um menor desempenho. Concomitantemente à proposta de trabalho futuro anterior, pode ser interessante verificar se adaptar o DCIA para lidar com cada característica específica das bases de dados resulta em uma melhor geração de protótipos;

- Neste trabalho os testes e comparações estatísticos foram conduzidos usando a métrica MAUC. Entretanto, existem outras métricas que comunicam diferentes informações sobre os desempenhos obtidos. Uma análise aprofundada dos desempenhos, levando-se em conta os resultados obtidos a partir de várias métricas, tem o potencial de produzir um conhecimento mais avançado. Tal conhecimento estará relacionado ao comportamento dos classificadores em relação às diferentes abordagens sobre o problema de desbalanceamento;

- A junção dos algoritmos de busca CSO, PSO e SGSA com o DCIA apresentaram vantagens e desvantagens. As três versões produzem, em geral, desempenho similar. É interessante, portanto, encontrar alguma maneira de se determinar qual algoritmo de busca melhor se encaixa no algoritmo, além de tentar mitigar quaisquer de suas desvantagens. Deve fazer parte desta pesquisa futura a combinação do DCIA com outros algoritmos de busca existentes. Além disso, é necessário também entender como o DCIA e os algoritmos de busca podem se beneficiar mutuamente na produção de um melhor desempenho;

- A métrica AUCarea é utilizada durante a Seleção de Atributos. Posteriormente, a métrica MAUC é utilizada para validar a geração de protótipos. Foi observado

que a métrica AUCarea algumas vezes não produz um bom desempenho quando utilizada também na geração de protótipos. Entretanto, existem outras métricas que podem ser testadas e combinadas. Portanto, é interessante verificar se existe outra combinação de métricas que possa produzir um melhor desempenho;

- O ajuste dos protótipos é realizado com um algoritmo de busca. Neste trabalho, foram utilizados CSO, GSA, PSO e SGSA. Entretanto, existem outros algoritmos de busca que podem ser investigados. Ainda, pode ser interessante a investigação de outros algoritmos de otimização nesta fase. Ao mesmo tempo, a adição de novos protótipos pode também obedecer a outro algoritmo. Uma proposta de trabalho futuro neste ponto é a utilização de *Growing Neural Gas Network* (FRITZKE, 1994), uma vez que tal algoritmo explicita as relações topológicas de uma distribuição. Desta forma, a adição de novos protótipos pode ser guiada e custar menos tempo ou ciclos;

- A Seleção de Atributos foi investigada utilizando a abordagem *wrapper* (MOAYEDIKIA et al., 2017). Dos algoritmos testados, o CSO como sugerido em (GU; CHENG; JIN, 2018) obteve o melhor resultado. Entretanto, existem alternativas ao *wrapper*, i.e., abordagens *filter* e híbrida;

- Como observado no Capítulo 4, os protótipos gerados pelo DCIA possuem um efeito positivo maior sobre o classificador 1NN. A razão para isto é que o 1NN é o classificador utilizado para validar a geração dos protótipos. A utilização de outros classificadores, ou até mesmo comitês de classificadores para validar a geração de novos protótipos pode gerar um conjunto de treinamento diferente. Este novo conjunto pode ter como efeito a melhoria do desempenho obtidos por outros classificadores além do 1NN;

- A utilização da técnica *5-fold SCV* está se tornando bastante comum no problema de classes desbalanceadas. A razão para tal é a quantidade de classes que possuem poucas instâncias. Entretanto, na literatura o mais comum é a utilização do *10-fold*, como indicado em (KOHAVI, 1995). Contudo, ainda existem classes com menos de cinco instâncias, as quais acabam por não estarem presentes em alguns dos conjuntos de testes organizados. A sobreamostragem de tais classes para que tenham tantas instâncias quanto o número de *folds* é uma possível solução. Todavia, é interessante a existência de um estudo mais aprofundado para demonstrar se a utilização de *hold-out* deve ou não ser considerada em tais casos, como sugerido em (SILVA; ZANCHETTIN, 2016a);

- Redes Neurais Profundas têm atraído bastante atenção da comunidade científica devido à qualidade dos resultados reportados. Sobre o problema de desbalanceamento, o *Deep Learning* tem sido mais utilizado em bases de imagens (JOHNSON;

KHOSHGOFTAAR, 2019). Entretanto, as arquiteturas até então utilizadas podem ser adaptadas para outros tipos de dados. Um trabalho futuro neste sentido, pode consistir em utilizar alguma arquitetura de *Deep Learning* como o classificador base para o DCIA, em vez do 1NN.

REFERÊNCIAS

ABDI, L.; HASHEMI, S. To combat multi-class imbalanced problems by means of over-sampling techniques. *IEEE Trans. Knowl. Data Eng.*, v. 28, n. 1, p. 238–251, 2016. Disponível em: <https://doi.org/10.1109/TKDE.2015.2458858>.

ALCALA-FDEZ, J.; FERNáNDEZ, A.; LUENGO, J.; DERRAC, J.; GARC'IA, S.; SANCHEZ, L.; HERRERA, F. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, v. 17, n. 2-3, p. 255–287, 2010.

ANAND, A.; PUGALENTHI, G.; FOGEL, G. B.; SUGANTHAN, P. N. An approach for classification of highly imbalanced data using weighting and undersampling. *Amino Acids*, v. 39, n. 5, p. 1385–1391, 2010. ISSN 1438-2199. Disponível em: <http://dx.doi.org/10.1007/s00726-010-0595-2>.

ARMSTRONG, R.; S.V.SLADE; F.EPERJESI. An introduction to analysis of variance (anova) with special reference to data from clinical experiments in optometry. *Ophthalmic and Physiological Optics*, v. 20, n. 3, p. 235–241, 2000.

AURENHAMMER, F. Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 23, n. 3, p. 345–405, sep 1991. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/116873.116880>.

BATISTA, G. E. de A. P. A. *Pré-processamento de Dados em Aprendizado de Máquina Supervisionado*. Tese (Doutorado) — Universidade de São Paulo - USP, São Paulo, mar 2003.

BHATIA, N.; VANDANA. Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security (IJCSIS)*, v. 8, n. 2, p. 302–305, 2010. ISSN 1947-5500.

BI, J.; ZHANG, C. An empirical comparison on state-of-the-art multi-class imbalance learning algorithms and a new diversified ensemble learning scheme. *Knowl.-Based Syst.*, v. 158, p. 81–93, 2018. Disponível em: <https://doi.org/10.1016/j.knosys.2018.05.037>.

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

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. *Classification and Regression Trees*. [S.l.]: Wadsworth, 1984. ISBN 0-534-98053-8.

BRODERSEN, K. H.; ONG, C. S.; STEPHAN, K. E.; BUHMANN, J. M. The balanced accuracy and its posterior distribution. In: *Proceedings of the 2010 20th International Conference on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2010. (ICPR '10), p. 3121–3124. ISBN 978-0-7695-4109-9. Disponível em: <http://dx.doi.org/10.1109/ICPR.2010.764>.

BUNKHUMPORNPAT, C.; SINAPIROMSARAN, K.; LURSINSAP, C. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In: *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. Springer-Verlag, 2009. (PAKDD '09, v. 5476), p. 475–482. ISBN 978-3-642-01306-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-01307-2_43>.

CHANG, C.-L. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, C-23, n. 11, p. 1179–1184, 1974. ISSN 0018-9340. DOI:10.1109/T-C.1974.223827.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, AI Access Foundation, USA, v. 16, n. 1, p. 321–357, jun. 2002. ISSN 1076-9757. Disponível em: <http://dl.acm.org/citation.cfm?id=1622407.1622416>.

CHAWLA, N. V.; JAPKOWICZ, N.; KOTCZ, A. Editorial: Special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, v. 6, n. 1, p. 1–6, jun 2004. ISSN 1931-0145. DOI:10.1145/1007730.1007733.

CHAWLA, N. V.; LAZAREVIC, A.; HALL, L. O.; BOWYER, K. W. Smoteboost: improving prediction of the minority class in boosting. In: *In Proceedings of the Principles of Knowledge Discovery in Databases, PKDD-2003*. [S.l.: s.n.], 2003. p. 107–119.

CHENG, R.; JIN, Y. A competitive swarm optimizer for large scale optimization. *IEEE Trans. Cybernetics*, v. 45, n. 2, p. 191–204, 2015.

CIESLAK, D. A.; CHAWLA, N. V. Learning decision trees for unbalanced data. In: *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I*. Berlin, Heidelberg: Springer-Verlag, 2008. (ECML PKDD '08), p. 241–256. ISBN 978-3-540-87478-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-87479-9_34>.

COHEN, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, v. 20, n. 1, p. 37–46, 1960.

COHEN, P. R. *Empirical Methods for Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1995. ISBN 0-262-03225-2.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, JMLR.org, v. 7, p. 1–30, dez. 2006. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=1248547.1248548>.

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

DIETTERICH, T. G.; BAKIRI, G. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In: *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*. AAAI Press, 1991. (AAAI'91), p. 572–577. ISBN 0-262-51059-6. Disponível em: <http://dl.acm.org/citation.cfm?id=1865756.1865765>.

EMARY, E.; ZAWBAA, H. M.; HASSANIEN, A. E. Binary ant lion approaches for feature selection. *Neurocomputing*, v. 213, p. 54–65, 2016.

ESTABROOKS, A.; JO, T.; JAPKOWICZ, N. A Multiple Resampling Method for Learning from Imbalanced Data Sets. *Computational Intelligence*, Blackwell Publishing, Inc., v. 20, n. 1, p. 18–36, fev. 2004. ISSN 1467-8640. Disponível em: <http://dx.doi.org/10.1111/j.0824-7935.2004.t01-1-00228.x>.

FARQUAD, M. A. H.; BOSE, I. Preprocessing unbalanced data using support vector machine. *Decis. Support Syst.*, Elsevier Science Publishers B. V., v. 53, n. 1, p. 226–233, apr 2012. ISSN 0167-9236.

FAYED, H. A.; HASHEM, S. R.; ATIYA, A. F. Self-generating prototypes for pattern classification. *Pattern Recognition*, Elsevier Science Inc., New York, NY, USA, v. 40, n. 5, p. 1498–1509, maio 2007. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2006.10.018>.

FERNÁNDEZ, A.; LÓPEZ, V.; GALAR, M.; JESUS, M. J. del; HERRERA, F. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-Based Systems*, v. 42, p. 97–110, apr 2013. DOI:10.1016/j.knosys.2013.01.018.

FERNÁNDEZ-NAVARRO, F.; HERVÁS-MARTÍNEZ, C.; GUTIÉRREZ, P. A. A dynamic over-sampling procedure based on sensitivity for multi-class problems. *Pattern Recognition*, v. 44, p. 1821—-1833, 2011.

FREEMAN, J. A. S.; SAAD, D. Learning and generalization in radial basis function networks. *Neural Computation*, MIT Press, Cambridge, MA, USA, v. 7, n. 5, p. 1000–1020, set. 1995. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1995.7.5.1000>.

FREUND, Y.; SCHAPIRE, R. E. Experiments with a new boosting algorithm. In: *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*. [S.l.: s.n.], 1996. p. 148–156.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, Academic Press, Inc., Orlando, FL, USA, v. 55, n. 1, p. 119–139, ago. 1997. ISSN 0022-0000. Disponível em: <http://dx.doi.org/10.1006/jcss.1997.1504>.

FRITZKE, B. A growing neural gas network learns topologies. In: *Proceedings of the 7th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1994. (NIPS'94), p. 625–632. Disponível em: <http://dl.acm.org/citation.cfm?id=2998687.2998765>.

F.R.S., K. P. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor Francis, v. 2, n. 11, p. 559–572, 1901. Disponível em: <https://doi.org/10.1080/14786440109462720>.

GALAR, M.; FERNáNDEZ, A.; BARRENECHEA, E.; BUSTINCE, H.; HERRERA, F. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recogn.*, Elsevier

Science Inc., New York, NY, USA, v. 44, n. 8, p. 1761–1776, ago. 2011. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2011.01.017>.

GALAR, M.; FERNáNDEZ, A.; BARRENECHEA, E.; BUSTINCE, H.; HERRERA, F. Dynamic classifier selection for one-vs-one strategy: Avoiding non-competent classifiers. *Pattern Recogn.*, Elsevier Science Inc., New York, NY, USA, v. 46, n. 12, p. 3412–3424, dez. 2013. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2013.04.018>.

GARCÍA, V.; MOLLINEDA, R. A.; SANCHEZ, J. S. Theoretical analysis of a performance measure for imbalanced data. *2010 20th International Conference on Pattern Recognition (ICPR)*, p. 617–620, 2010. ISSN 1051-4651. DOI:10.1109/ICPR.2010.156.

GARCÍA, V.; SÁNCHEZ, J. S.; MOLLINEDA, R. A. On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowl.-Based Syst.*, v. 25, n. 1, p. 13–21, 2012. DOI:10.1016/j.knosys.2011.06.013.

GARCÍA, V.; SÁNCHEZ, J. S.; MOLLINEDA, R. A.; ALEJO, R.; SOTOCA, J. M. The class imbalance problem in pattern classification and learning. 2009.

GEEM, Z. W. Novel derivative of harmony search algorithm for discrete design variables. *Applied Mathematics and Computation*, v. 199, n. 1, p. 223–230, 2008.

GOWDA, K.; KRISHNA, G. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory*, v. 25, p. 488–490, 1979.

GU, S.; CHENG, R.; JIN, Y. Feature selection for high-dimensional classification using a competitive swarm optimizer. *Soft Comput.*, v. 22, n. 3, p. 811–822, 2018.

HAIXIANG, G.; YIJING, L.; SHANG, J.; MINGYUN, G.; YUANYUE, H.; BING, G. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems With Applications*, v. 73, p. 220–239, 2017.

HAN, H.; WANG, W.-Y.; MAO, B.-H. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In: *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I*. Berlin, Heidelberg: Springer-Verlag, 2005. (ICIC'05), p. 878–887. ISBN 3-540-28226-2, 978-3-540-28226-6. Disponível em: <http://dx.doi.org/10.1007/11538059_91>.

HAN, J.; KAMBER, M.; PEI, J. *Data mining concepts and techniques, third edition*. Waltham, Mass.: Morgan Kaufmann Publishers, 2012. ISBN 0123814790.

HAND, D. J.; TILL, R. J. A simple generalisation of the area under the roc curve for multiple class classification problems. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 45, n. 2, p. 171–186, out. 2001. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A:1010920819831>.

HASSAN, M. R.; RAMAMOHANARAO, K.; KARMAKAR, C. K.; HOSSAIN, M. M.; BAILEY, J. A novel scalable multi-class ROC for effective visualization and computation. In: ZAKI, M. J.; YU, J. X.; RAVINDRAN, B.; PUDI, V. (Ed.). *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I*. Springer, 2010. (Lecture Notes

in Computer Science, v. 6118), p. 107–120. ISBN 978-3-642-13656-6. Disponível em: <https://doi.org/10.1007/978-3-642-13657-3_14>.

HASTIE, T.; TIBSHIRANI, R. Classification by pairwise coupling. *Ann. Statist.*, v. 26, n. 2, p. 451–471, 1998.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. [S.l.]: Prentice Hall, 1999.

HE, H.; BAI, Y.; GARCIA, E. A.; LI, S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, p. 1322–1328, 2008. DOI:10.1109/IJCNN.2008.4633969.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

HU, B.-G.; DONG, W.-M. A study on cost behaviors of binary classification measures in class-imbalanced problems. *CoRR*, abs/1403.7100, 2014. Disponível em: <http://arxiv.org/abs/1403.7100>.

HU, S.; LIANG, Y.; MA, L.; HE, Y. Msmote: Improving classification performance when training data is imbalanced. *Second International Workshop on Computer Science and Engineering, WCSE'09*, v. 2, p. 13–17, 2009. DOI:10.1109/WCSE.2009.756.

ISLAM, M. M.; YAO, X.; NIRJON, S. M. S.; ISLAM, M. A.; MURASE, K. Bagging and boosting negatively correlated neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE Press, Piscataway, NJ, USA, v. 38, n. 3, p. 771–784, jun. 2008. ISSN 1083-4419. Disponível em: <https://doi.org/10.1109/TSMCB.2008.922055>.

JAIN, Y. K.; BHANDARE, S. K. Min max normalization based data perturbation method for privacy protection. *International Journal of Computer Communication Technology*, v. 2, p. 45–50, 2011.

JAPKOWICZ, N.; STEPHEN, S. The class imbalance problem: A systematic study. *Intell. Data Anal.*, IOS Press, v. 6, n. 5, p. 429–449, oct 2002. ISSN 1088-467X.

JOHNSON, J. M.; KHOSHGOFTAAR, T. M. Survey on deep learning with class imbalance. *Journal of Big Data*, v. 6, n. 27, p. 1–54, 2019. ISSN 2196-1115. Disponível em: <https://doi.org/10.1186/s40537-019-0192-5>.

KANG, S.; CHO, S.; KANG, P. Constructing a multi-class classifier using one-against-one approach with different binary classifiers. *Neurocomputing*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 149, n. PB, p. 677–682, fev. 2015. ISSN 0925-2312. Disponível em: <http://dx.doi.org/10.1016/j.neucom.2014.08.006>.

KANNAN, S. S.; RAMARAJ, N. A novel hybrid feature selection via symmetrical uncertainty ranking based local memetic search algorithm. *Knowledge-Based Systems*, v. 23, n. 6, p. 580–585, 2010.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KOÇO, S.; CAPPONI, C. On multi-class classification through the minimization of the confusion matrix norm. In: ONG, C. S.; HO, T. B. (Ed.). *ACML*. [S.l.]: JMLR.org, 2013. (JMLR Workshop and Conference Proceedings, v. 29), p. 277–292.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (IJCAI'95), p. 1137–1143. ISBN 1-55860-363-8. Disponível em: <http://dl.acm.org/citation.cfm?id=1643031.1643047>.

KRUSKAL, W.; WALLIS, W. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, v. 47, p. 583–621, 1952.

KRUSKAL-WALLIS Test. In: THE Concise Encyclopedia of Statistics. New York, NY: Springer New York, 2008. p. 288–290. ISBN 978-0-387-32833-1. Disponível em: <https://doi.org/10.1007/978-0-387-32833-1_216>.

KUBAT, M.; MATWIN, S. Addressing the curse of imbalanced training sets: One-sided selection. In: *In Proceedings of the Fourteenth International Conference on Machine Learning*. [S.l.]: Morgan Kaufmann, 1997. p. 179–186.

LAURIKKALA, J. Improving identification of difficult small classes by balancing class distribution. In: *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine*. London, UK, UK: Springer-Verlag, 2001. (AIME '01), p. 63–66. ISBN 3-540-42294-3.

LEE, C.-Y.; LEE, Z.-J. A novel algorithm applied to classify unbalanced data. *Appl. Soft Comput.*, v. 12, n. 8, p. 2481–2485, 2012. ISSN 1568-4946. DOI:10.1016/j.asoc.2012.03.051.

LI, X.; WANG, L.; ; SUNG, E. Improving adaboost for classification on small training sample sets with active learning. In: *The Sixth Asian Conference on Computer Vision ACCV*. Korea: [s.n.], 2004.

LIN, H.-Y. Efficient classifiers for multi-class classification problems. *Decision Support Systems*, v. 53, p. 473–481, 2012.

LING, C. X.; HUANG, J.; ZHANG, H. Auc: A statistically consistent and more discriminating measure than accuracy. In: *IJCAI'03 Proceedings of the 18th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. p. 519–524. Disponível em: <http://dl.acm.org/citation.cfm?id=1630659.1630736>.

LING, C. X.; SHENG, V. S. Cost-sensitive learning and the class imbalance problem. *Encyclopedia of Machine Learning*, p. 231–235, 2008.

LING, C. X.; YANG, Q.; WANG, J.; ZHANG, S. Decision trees with minimal costs. In: BRODLEY, C. E. (Ed.). *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. ACM, 2004. (ACM International Conference Proceeding Series, v. 69). Disponível em: <https://doi.org/10.1145/1015330.1015369>.

LIU, Y.; AN, A.; HUANG, X. Boosting prediction accuracy on imbalanced datasets with svm ensembles. In: *Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer-Verlag, 2006. (PAKDD'06), p. 107–118. ISBN 3-540-33206-5, 978-3-540-33206-0. Disponível em: <http://dx.doi.org/10.1007/11731139_15>.

LIU, Y.; YAO, X. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE Press, Piscataway, NJ, USA, v. 29, n. 6, p. 716–725, dez. 1999. ISSN 1083-4419. Disponível em: <https://doi.org/10.1109/3477.809027>.

LÓPEZ, V.; FERNÁNDEZ, A.; GARCÍA, S.; PALADE, V.; HERRERA, F. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, v. 250, p. 113–141, 2013. DOI:10.1016/j.ins.2013.07.007.

LÓPEZ, V.; FERNÁNDEZ, A.; HERRERA, F. On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Information Sciences*, v. 257, p. 1–13, 2014.

LÓPEZ, V.; TRIGUERO, I.; CARMONA, C. J.; GARCÍA, S.; HERRERA, F. Addressing imbalanced classification with instance generation techniques: IPADE-ID. *Neurocomputing*, v. 126, p. 15–28, 2014. Disponível em: <https://doi.org/10.1016/j.neucom.2013.01.050>.

MAATEN, L. van der; HINTON, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, v. 9, p. 2579–2605, 2008.

MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In: *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*. [S.l.: s.n.], 1967. p. 281–297.

MAFARJA, M.; MIRJALILI, S. Whale optimization approchaes for wrapper feature selection. *Applied Soft Computing*, v. 62, p. 441–453, 2018.

MAHALANOBIS, P. C. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, v. 2, p. 49–55, 1936.

MALDONADO, S.; MONTECINOS, C. Robust classification of imbalanced data using one-class and two-class svm-based multiclassifiers. *Intell. Data Anal.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 18, n. 1, p. 95–112, jan. 2014. ISSN 1088-467X. DOI:10.3233/IDA-130630. Disponível em: <http://dx.doi.org/10.3233/IDA-130630>.

MALLENAHALLI, N.; SARMA, T. H. A tunable particle swarm size optimization algorithm for feature selection. *CoRR*, p. 1–7, 2018. Disponível em: <http://arxiv.org/abs/1806.10551>.

MILLÁN-GIRALDO, M.; GARCÍA, V.; SÁNCHEZ, J. S. Prototype selection in imbalanced data for dissimilarity representation - A preliminary study. In: CARMONA, P. L.; SÁNCHEZ, J. S.; FRED, A. L. N. (Ed.). *ICPRAM 2012 - Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods, Volume 1,*

Vilamoura, Algarve, Portugal, 6-8 February, 2012. [S.l.]: SciTePress, 2012. p. 242–247. ISBN 978-989-8425-98-0.

MIRJALILI, S.; LEWIS, A. The whale optimization algorithm. *Advances in Engineering Software*, v. 95, n. C, p. 51–67, 2016.

MOAYEDIKIA, A.; ONG, K.; BOO, Y. L.; YEOH, W. G. S.; JENSEN, R. Feature selection for high dimensional imbalanced class data using harmony search. *Engineering Applications of Artificial Intelligence*, v. 57, p. 38–49, 2017.

MOLLINEDA, R. A.; SÁNCHEZ, J. S.; SOTOCA, J. M. Data characterization for effective prototype selection. In: *Proc. of the 2nd Iberian Conf. on Pattern Recognition and Image Analysis*. [S.l.]: Springer, 2005. p. 27–34.

MORENO-TORRES, J. G.; RAEDER, T.; ALAIZ-RODRÍGUEZ, R.; CHAWLA, N. V.; HERRERA, F. A unifying view on dataset shift in classification. *Pattern Recognition*, Elsevier Science Inc., New York, NY, USA, v. 45, n. 1, p. 521–530, jan. 2012. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2011.06.019>.

MORENO-TORRES, J. G.; SÁEZ, J. A.; HERRERA, F. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Transatcions On Neural Networks and Learning Systems*, v. 23, n. 8, p. 1304–1312, 2012.

MOSCATO, P.; COTTA, C. A gentle introduction to memetic algorithms. *Handbook of Metaheuristics, International Series in Operations Research and Management Science*, Springer, v. 57, p. 105–144, 2003.

MOSLEY, L. *A balanced approach to the multi-class imbalance problem*. Tese (Doutorado) — Iowa State University, Iowa, 2013.

MUKHERJEE, I.; SCHAPIRE, R. E. A theory of multiclass boosting. *Journal of Machine Learning Research*, v. 14, n. 1, p. 437–497, 2013.

NAPIERALA, K.; STEFANOWSKI, J. BRACID: a comprehensive approach to learning rules from imbalanced data. *J. Intell. Inf. Syst.*, v. 39, n. 2, p. 335–373, 2012. Disponível em: <http://dx.doi.org/10.1007/s10844-011-0193-0>.

NAPIERALA, K.; STEFANOWSKI, J.; WILK, S. Learning from imbalanced data in presence of noisy and borderline examples. In: SZCZUKA, M. S.; KRYSZKIEWICZ, M.; RAMANNA, S.; JENSEN, R.; HU, Q. (Ed.). *Rough Sets and Current Trends in Computing - 7th International Conference, RSCTC 2010, Warsaw, Poland, June 28-30,2010. Proceedings*. Springer, 2010. (Lecture Notes in Computer Science, v. 6086), p. 158–167. ISBN 978-3-642-13528-6. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13529-3_18>.

NGUYEN, G. H.; BOUZERDOUM, A.; PHUNG, S. L. Learning pattern classification tasks with imbalanced data sets. *In P. Yin (Eds.), Pattern recognition*, p. 193–208, 2009.

OH, S.-H. Error back-propagation algorithm for classification of imbalanced data. *Neurocomputing*, v. 74, n. 6, p. 1058–1061, 2011.

OLIVEIRA, D. V. R.; aES, G. R. M.; CAVALCANTI, G. D. C.; REN, T. I. Improved self-generating prototypes algorithm for imbalanced datasets. In: *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*. [S.l.]: IEEE Computer Society, 2012. p. 904–909.

OLIVEIRA, D. V. R.; CAVALCANTI, G. D. C.; REN, T. I.; SILVA, R. M. A. Evolutionary adaptive self-generating prototypes for imbalanced datasets. In: *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*. [S.l.]: IEEE, 2015. p. 1–8. ISBN 978-1-4799-1960-4.

ORTIGOSA-HERNáNDEZ, J.; INZA, I. naki; LOZANO, J. A. Measuring the class-imbalance extent of multi-class problems. *Pattern Recognition Letters*, Elsevier Science Inc., New York, NY, USA, v. 98, n. C, p. 32–38, out. 2017. ISSN 0167-8655. Disponível em: <https://doi.org/10.1016/j.patrec.2017.08.002>.

OU, G.; MURPHEY, Y. L. Multi-class pattern classification using neural networks. *Pattern Recognition*, v. 40, n. 1, p. 4–18, 2007. DOI:10.1016/j.patcog.2006.04.041.

OUGIAROGLOU, S.; EVANGELIDIS, G. Fast and accurate k-nearest neighbor classification using prototype selection by clustering. *2012 16th Panhellenic Conference on Informatics (PCI)*, p. 168–173, 2012. DOI:10.1109/PCi.2012.69.

PRATI, R. C.; BATISTA, G. E. A. P. A.; MONARD, M. C. Class imbalances versus class overlapping: An analysis of a learning system behavior. In: MONROY, R.; ARROYO-FIGUEROA, G.; SUCAR, L. E.; AZUELA, J. H. S. (Ed.). *Third Mexican International Conference on Artificial Intelligence (MICAI): Advances in Artificial Intelligence*. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v. 2972), p. 312 – 321. ISBN 3-540-21459-3.

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

RANAWANA, R.; PALADE, V. Optimized precision - a new measure for classifier performance evaluation. *2006 IEEE International Conference on Evolutionary Computation*, p. 2254–2261, 2006. ISSN 1089-778X. DOI:10.1109/CEC.2006.1688586.

RASHEDI, E.; NEZAMABADI-POUR, H.; SARYAZDI, S. GSA: A gravitational search algorithm. *Information Sciences*, v. 179, p. 2232 – 2248, 2009.

REZAEI, M.; NEZAMABADI-POUR, H. Using gravitational search algorithm in prototype generation for nearest neighbor classification. *Neurocomputing*, v. 157, p. 256–263, 2015.

RUIZ, R.; RIQUELME, J. C.; AGUILAR-RUIZ, J. S.; GARCÍA-TORRES, M. Fast feature selection aimed at high-dimensional data via hybrid-sequential-ranked searches. *Expert Syst. Appl.*, v. 39, n. 12, p. 11094–11102, 2012.

SÁNCHEZ-CRISOSTOMO, J. P.; ALEJO, R.; LÓPEZ-GONZÁLEZ, E.; VALDOVINOS, R. M.; PACHECO-SÁNCHEZ, J. H. Empirical analysis of assessments metrics for multi-class imbalance learning on the back-propagation context. In: TAN, Y.; SHI, Y.; COELLO, C. A. C. (Ed.). *Advances in Swarm Intelligence - 5th International Conference, ICSI 2014, Hefei, China, October 17-20, 2014, Proceedings, Part II*. Springer, 2014.

(Lecture Notes in Computer Science, v. 8795), p. 17–23. ISBN 978-3-319-11896-3. Disponível em: <https://doi.org/10.1007/978-3-319-11897-0_3>.

SILVA, E. J. R.; ZANCHETTIN, C. On the existence of a threshold in class imbalance problems. *IEEE International Conference on Systems, Man, and Cybernetics, 2015, Hong Kong*, p. 2714–2719, 2015. DOI:10.1109/SMC.2015.474.

SILVA, E. J. R.; ZANCHETTIN, C. On validation setup for multiclass imbalanced data sets. *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, p. 468–473, 2016. Disponível em: <https://doi.org/10.1109/BRACIS.2016.090>.

SILVA, E. J. R.; ZANCHETTIN, C. A voronoi diagram based classifier for multiclass imbalanced data sets. *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, p. 109 – 114, 2016. Disponível em: <http://doi.org/10.1109/BRACIS.2016.030>.

SILVA, E. J. R.; ZANCHETTIN, C. Dynamic centroid insertion and adjustment for data sets with multiple imbalanced classes. In: TETKO, I. V.; KŮRKOVÁ, V.; KARPOV, P.; THEIS, F. (Ed.). *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*. Cham: Springer International Publishing, 2019. p. 766–778. ISBN 978-3-030-30484-3. Disponível em: <https://doi.org/10.1007/978-3-030-30484-3_60>.

SNEDECOR, G.; COCHRAN, W. *Statistical Methods*. 7th. ed. Ames: Iowa State University Press, 1980.

STEFANOWSKI, J.; WILK, S. Selective pre-processing of imbalanced data for improving classification performance. In: *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*. Berlin, Heidelberg: Springer-Verlag, 2008. (DaWaK '08), p. 283–292. ISBN 978-3-540-85835-5. Disponível em: <http://dx.doi.org/10.1007/978-3-540-85836-2_27>.

STORN, R.; PRICE, K. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, Kluwer Academic Publishers, Hingham, MA, USA, v. 11, n. 4, p. 341–359, dez. 1997. ISSN 0925-5001. Disponível em: <https://doi.org/10.1023/A:1008202821328>.

SUN, Y.; KAMEL, M. S.; WANG, Y. Boosting for learning multiple classes with imbalanced class distribution. In: *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*. IEEE Computer Society, 2006. p. 592–602. ISBN 0-7695-2701-9. Disponível em: <https://doi.org/10.1109/ICDM.2006.29>.

SUN, Y.; WONG, A. K. C.; WANG, Y. Parameter inference of cost-sensitive boosting algorithms. In: *Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*. Berlin, Heidelberg: Springer-Verlag, 2005. (MLDM'05), p. 21–30. ISBN 3-540-26923-1, 978-3-540-26923-6. Disponível em: <http://dx.doi.org/10.1007/11510888_3>.

SUN, Z.; SONG, Q.; ZHU, X.; SUN, H.; XU, B.; ZHOU, Y. A novel ensemble method for classifying imbalanced data. *Pattern Recogn.*, Elsevier Science Inc., New York, NY, USA, v. 48, n. 5, p. 1623–1637, maio 2015. ISSN 0031-3203. Disponível em: <http://dx.doi.org/10.1016/j.patcog.2014.11.014>.

TANG, Y.; ZHANG, Y.-Q.; CHAWLA, N. V.; KRASSER, S. Svms modeling for highly imbalanced classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 39, n. 1, p. 281–288, 2009. ISSN 1083-4419. DOI:10.1109/TSMCB.2008.2002909.

TOMEK, I. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics (SMC-6)*, p. 769 – 772, 1976. DOI:10.1109/TSMC.1976.4309452.

TRIGUERO, I.; GARCÍA, S.; HERRERA, F. IPADE: iterative prototype adjustment for nearest neighbor classification. *IEEE Trans. Neural Networks*, v. 21, n. 12, p. 1984–1990, 2010.

TURNEY, P. D. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research 2*, p. 369–409, 1995.

VERBIEST, N.; RAMENTOL, E.; CORNELIS, C.; HERRERA, F. Improving SMOTE with fuzzy rough prototype selection to detect noise in imbalanced classification data. In: PAVÓN, J.; DUQUE-MÉNDEZ, N. D.; FUENTES-FERNÁNDEZ, R. (Ed.). *Advances in Artificial Intelligence - IBERAMIA 2012 - 13th Ibero-American Conference on AI, Cartagena de Indias, Colombia, November 13-16, 2012. Proceedings*. [S.l.]: Springer, 2012. (Lecture Notes in Computer Science, v. 7637), p. 169–178. ISBN 978-3-642-34653-8.

WANG, Q. A hybrid sampling svm approach to imbalanced data classification. *Abstract and Applied Analysis*, v. 2014, 2014. Article ID 972786, 7 pages, doi:10.1155/2014/972786.

WANG, S.; YAO, X. Multiclass imbalance problems: Analysis and potential solutions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, v. 42, n. 4, p. 1119–1130, 2012.

WEISS, G.; PROVOST, F. *The Effect of Class Distribution on Classifier Learning: An Empirical Study*. [S.l.], 2001.

WILCOXON, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, v. 1, n. 6, p. 80 – 83, 1945.

YANG, W.; WANG, K.; ZUO, W. Neighborhood component feature selection for high-dimensional data. *Journal of Computers*, v. 7, n. 1, p. 161–168, 2012.

YANG, X.; KUANG, Q.; ZHANG, W.; ZHANG, G. AMDO: an over-sampling technique for multi-class imbalanced problems. *IEEE Trans. Knowl. Data Eng.*, v. 30, n. 9, p. 1672–1685, 2018. Disponível em: <https://doi.org/10.1109/TKDE.2017.2761347>.

YEN, S.-J.; LEE, Y.-S.; LIN, C.-H.; YING, J.-C. Investigating the effect of sampling methods for imbalanced data distributions. *2006 IEEE International Conference on Systems, Man and Cybernetics*, v. 5, p. 4163–4168, 2006. DOI:10.1109/ICSMC.2006.384787.

YIJING, L.; HAIXIANG, G.; XIAO, L.; YANAN, L.; JINLING, L. Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data. *Knowledge-Based Systems*, v. 94, p. 88–104, 2016.

ZENG, X.; MARTINEZ, T. R. Distribution-balanced stratified cross-validation for accuracy estimation. *Journal of Experimental and Theoretical Artificial Intelligence*, v. 12, n. 1, p. 1–12, 2000.

ZHANG, H.; LI, M. Rwo-sampling: A random walk over-sampling approach to imbalanced data classification. *Information Fusion*, v. 20, p. 99–116, 2014. DOI:10.1016/j.inffus.2013.12.003.

ZONG, W.; HUANG, G.-B.; CHEN, Y. Weighted extreme learning machine for imbalance learning. *Neurocomputing*, v. 101, p. 229–242, 2013.

APÊNDICE A – DETALHES DAS MODIFICAÇÕES DO VDBC

Neste apêndice todas as modificações realizadas sobre o VDBC são apresentadas em detalhes. Cada seção consiste na apresentação de uma das modificações, juntamente aos resultados obtidos e respectivas análises.

A.1 VDBC.M1

A primeira modificação do VDBC, chamada VDBC.M1, tem por objetivo verificar a geração de protótipos ao se observar k vizinhos diferentes, com $k = 1, ..., 5$. Com a evidência de que um maior desbalanceamento faz com que o espaço amostral seja mais complexo, valores diferentes de k podem fazer com que mais protótipos sejam criados em regiões mais complexas, e menos protótipos sejam criados em regiões mais simples do espaço amostral. Ainda existe a possibilidade das instâncias das regiões mais fáceis serem mais rapidamente reduzidas em seus respectivos protótipos.

Nesta modificação, o modelo de geração de protótipos continua basicamente o mesmo, i.e., se o(s) vizinho(s) da instância selecionada for(em) da mesma classe, um protótipo é gerado; caso contrário, a instância corrente é transformada em protótipo. Entretanto, o algoritmo agora passa a verificar três casos distintos: (1) quanto todos os k vizinhos são protótipos, (2) os k vizinhos são protótipos e outras instâncias, e (3) todos os vizinhos são instâncias.

Quanto ao primeiro caso, se todos os protótipos forem da mesma classe, um centroide é criado entre os mesmos e substitui os já existentes. Isto significa que os protótipos que são vizinhos da instância corrente são excluídos do conjunto de protótipos, e o novo centroide criado é adicionado ao conjunto. Os outros dois casos já são verificados desde o algoritmo original e não sofreram mudanças.

Nas Tabelas 28 e 29 são apresentados os resultados obtidos. Na primeira tabela são apresentados as médias e desvios-padrões obtidos com a métrica MAUC. A segunda tabela apresenta as médias e desvios-padrões da quantidade de protótipos gerados.

Na Tabela 28 os valores evidenciados em **negrito** são os melhores resultados obtidos, de acordo com as comparações estatísticas. Quando mais de um valor é evidenciado na mesma linha, isto significa que os mesmos são estatisticamente equivalentes. Entretanto, se nenhum valor está evidenciado em uma linha, isto significa que todos os valores são estatisticamente equivalentes.

De início, o que chama atenção é a diferença entre o desempenho do VDBC e VDBC.M1 com 1NN. Apesar de ambos serem executados observando-se apenas o vizinho mais próximo, há uma diferença fundamental no algoritmo. No original, se o vizinho mais próximo for um protótipo da mesma classe nenhuma ação é tomada. Na modificação efetuada, este

Tabela 28 – Média e Desvio-Padrão do VDBC.M1 através da métrica MAUC.

Base	**1NN**	**2NN**	**3NN**	**4NN**	**5NN**
Abalone	0,7030±0,0213	0,7004±0,0193	0,6917±0,0170	0,6921±0,0130	0,6862±0,0202
Arrhythmia	0,7781±0,0376	0,7744±0,0375	0,7584±0,0229	0,7553±0,0374	0,7684±0,0395
Balance Scale	0,6408±0,0290	0,6391±0,0412	**0,7250**±0,0379	**0,7142**±0,0346	**0,7394**±0,0224
Car Evaluation	0,6962±0,0278	0,8215±0,0084	**0,8925**±0,0170	**0,8840**±0,0395	**0,8740**±0,0218
Contraceptive	**0,5985**±0,0218	0,5677±0,0201	0,5666±0,0032	**0,5940**±0,0369	**0,5808**±0,0358
Dermatology	0,6310±0,0294	0,6843±0,0308	0,7671±0,0393	**0,8247**±0,0523	**0,8587**±0,0601
E.coli	0,8197±0,0500	**0,8528**±0,0439	**0,9017**±0,0315	**0,9071**±0,0379	**0,8966**±0,0266
Gene	0,5328±0,0139	0,5755±0,0267	0,6154±0,0206	**0,6354**±0,0177	**0,6620**±0,0202
Glass	0,7104±0,0402	**0,8819**±0,0319	**0,8910**±0,0460	**0,8656**±0,0300	**0,8939**±0,0271
Hayes-Roth	0,5545±0,0226	0,5963±0,0490	**0,6676**±0,1396	**0,7425**±0,1190	**0,7563**±0,0414
Horse	0,5367±0,0235	0,5240±0,0121	0,5402±0,0180	0,5206±0,0098	0,5351±0,0167
Nursery	0,6616±0,0291	0,7530±0,0396	**0,7974**±0,0160	**0,8119**±0,0080	**0,8102**±0,0094
Page Blocks	**0,8356**±0,0198	0,7039±0,0238	0,7081±0,0317	0,7419±0,0345	0,7494±0,0302
Post Operative	0,5350±0,0251	**0,6678**±0,1366	**0,6170**±0,0549	**0,6134**±0,0819	**0,6729**±0,1028
Satimage	0,8810±0,0074	0,7570±0,0149	0,8869±0,0045	0,9219±0,0080	**0,9367**±0,0073
Shuttle	**0,8524**±0,0226	0,7856±0,0256	0,7581±0,0167	0,7693±0,0270	0,7252±0,0331
Soybean	0,7932±0,0234	0,8156±0,0390	0,9131±0,0135	**0,9485**±0,0108	**0,9644**±0,0140
Thyroid	0,5511±0,0164	0,5869±0,0175	0,6932±0,0179	**0,7349**±0,0230	**0,7463**±0,0242
Wine	**0,8783**±0,0451	0,6818±0,0715	**0,8480**±0,0352	**0,8325**±0,0644	**0,8106**±0,0731
Yeast	**0,7998**±0,0253	0,7008±0,0319	0,6648±0,0232	0,6912±0,0146	0,6665±0,0268
Zoo	0,7021±0,0499	**0,7598**±0,0923	**0,8248**±0,0906	**0,8806**±0,0417	**0,8448**±0,0595

caso é tratado diferentemente. Um centroide é criado entre a instância corrente e o protótipo. O novo centroide então substitui o protótipo que já existia. Este fator teve como efeito uma redução de desempenho entre VDBC e VDBC.M1 na maioria das bases de dados. Algumas vezes essa redução é drástica, caindo quase 30% percentuais. Em poucas bases o desempenho se manteve e em duas das bases (*Page-Blocks* e *Yeast*) houve uma melhora.

A melhora de desempenho do algoritmo em duas bases enquanto no restante há piora de desempenho, é uma evidência de que as características internas das bases, i.e., como as classes estão dispostas no espaço amostral, algumas vezes são complexas o bastante para que seja necessária uma abordagem mais específica. A geração de protótipos a partir de outros protótipos evidencia também que a redução de instâncias pode ser afetada pela posição dos protótipos gerados. Em outras palavras, há casos em que alguns protótipos são movidos no espaço por causa de alguma instância próxima de sua própria classe. Isto teve como efeito uma quantidade diferente de protótipos gerados em relação ao algoritmo original.

Observando-se novamente a Tabela 28 é possível notar que, de todas as variações, observar os cinco vizinhos mais próximos produz os melhores resultados. O VDBC.M1 com 5NN obteve o melhor desempenho em 18 das 21 bases de dados. A variação com 4NN conseguiu obter o melhor desempenho em 17 das 21 bases, empatando com a varição de 5NN em praticamente todas as bases, exceto na *Satimage*. Mas ainda que a diferença estatística tenha sido significativa, a diferença real foi pequena, ou seja, de 0,92 para 0,93.

Tabela 29 – Média e Desvio-Padrão da quantidade de protótipos gerados pela primeira modificação do VDBC.

Base	1NN	2NN	3NN	4NN	5NN
Abalone	1982±3,0332	1700±27,5409	1601±33,9293	1574±38,5136	1501±28,3602
Arrhythmia	183±5,8052	273±9,7826	287±6,0663	314±4,6690	320±4,9295
Balance Scale	431±3,3615	283±14,3248	316±9,1269	337±10,1833	363±18,5742
Car Evaluation	1230±3,7815	759±26,1285	818±16,8731	957±14,3073	1016±20,4768
Contraceptive	666±17,3695	857±17,4126	983± 8,4143	1040±10,7703	1094±10,8305
Dermatology	147±1,9494	161±10,8028	190±5,0200	204±19,1703	218±12,6174
E.coli	164±2,1679	189±9,3113	190±3,1145	206±10,6911	207±4,9699
Gene	797±19,3598	1574±19,3210	1983±14,5190	2194±70,1520	2315±58,4423
Glass	102±2,4083	115±6,4576	130±5,2154	136±5,2440	139±4,9699
Hayes-Roth	45±3,3615	71±3,5637	88±7,0922	110±8,7636	121±5,9582
Horse	169±1,6733	147±10,8259	155±10,6395	167±4,3243	155±11,3446
Nursery	9575±6,7602	5733±23,1776	5834±65,1291	6324±25,4657	6850±87,4157
Page Blocks	2652±17,3551	2918±5,7271	2932±63,8216	2914±21,1021	2945±41,0707
Post Operative	38±1,3038	56±4,2190	51±3,5355	62±8,0808	70±2,8284
Satimage	2432±9,1542	3449±12,2963	3539±23,1452	3693±65,0285	3753±36,6333
Shuttle	30763±49,5853	28668±84,0910	31435±72,5899	31676±61,6157	31817±135,950
Soybean	274±03,3615	342±7,0214	361±9,7826	359±9,8843	362±13,2853
Thyroid	3532±14,4049	3817±26,1285	3923±36,1981	3858±49,9530	3917±22,5942
Wine	90±3,3615	99±1,9235	100±3,2094	112±3,9370	109±4,0373
Yeast	706±6,9065	909±10,3537	975±9,8641	1053±10,4738	1082±14,5499
Zoo	34±1,5811	46±4,8477	50±4,7749	59±6,1400	60±3,8987

As demais variações não *acompanharam* o 4NN e 5NN, obtendo menos resultados entre os melhores obtidos.

A partir da Tabela 29 é possível verificar que as variações 3NN, 4NN e 5NN comumente geraram mais protótipos. Da forma como o conjunto de protótipos está sendo gerado, uma maior redução do espaço amostral resulta em desempenhos piores. Contudo, uma rápida observação sobre os resultados do VDBC e VDBC.M1 com 5NN evidencia que observar mais vizinhos, e com isso modificar o conjunto de protótipos, produz resultados similares ou piores aos que já haviam sido obtidos com o VDBC. Portanto, uma melhora do desempenho, aliado a uma maior redução do espaço amostral com VDBC deve ser feita seguindo outra estratégia.

A.2 VDBC.M2

A segunda modificação do VDBC é baseada na proposta do SMOTE (CHAWLA et al., 2002). No SMOTE, cada instância da classe minoritária cria uma ou mais instâncias sintéticas. Essas novas instâncias estarão posicionadas na linha entre a instância corrente e um de seus k vizinhos mais próximos.

No contexto de múltiplas classes, o conceito de classe minoritária se torna mais difícil, ou abrangente. Portanto, é necessária a utilização de alguma estratégia para selecionar qual ou quais classes serão consideradas as minoritárias. No VDBC.M2 a estratégia para

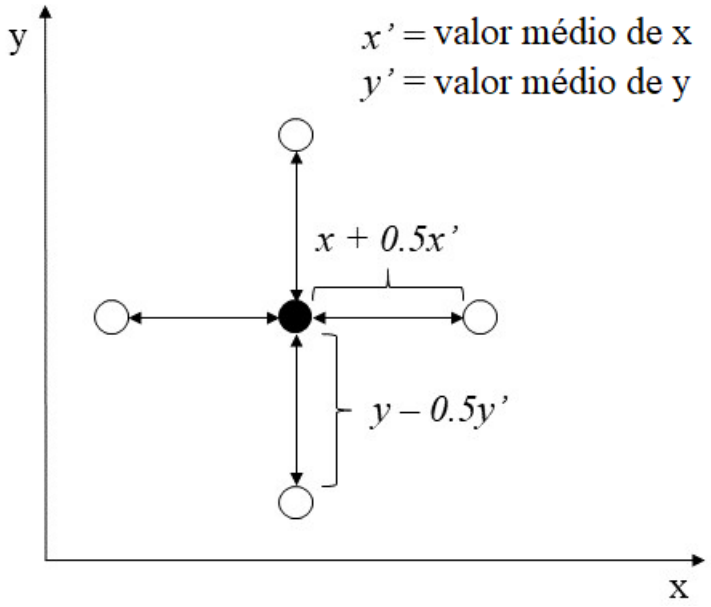

Figura 48 – Exemplo de sobreamostragem com um *singleton* e duas dimensões.

definir quais classes são minoritárias é a seguinte: (1) calcular a probabilidade *a priori* de cada classe; (2) calcular a média das probabilidades *a priori*; (3) todas as classes cujas probabilidades *a priori* sejam menores que a média, são consideradas minoritárias.

As probabilidades *a priori* são calculadas a partir do conjunto de treinamento. Seu cálculo corresponde à quantidade de instâncias de determinada classe dividida pela quantidade total de instâncias. Para melhor entendimento, levemos em consideração três classes com 100, 50 e 25 instâncias, respectivamente. A quantidade total de instâncias é 175. Logo, a probabilidade *a priori* das classes são $100/175 = 0,5714$, $50/175 = 0,2857$ e $25/175 = 0,1429$, respectivamente. A média das probabilidades é $0,3333$, portanto, as duas menores classes podem ser consideradas como minoritárias, de acordo com a estratégia adotada.

Uma vez selecionadas as classes minoritárias, o processo de sobreamostragem se dá da seguinte forma: Para cada par de instâncias, uma instância sintética é criada entre as mesmas. No caso de uma classe possuir um *singleton*, i.e., apenas uma única instância de treinamento, o processo de sobreamostragem é da seguinte forma: (1) calcular o valor médio de todas as instâncias para cada atributo/dimensão; (2) criar $2d$ novas instâncias, no qual d é a quantidade de dimensões. A posição das novas instâncias em cada dimensão é calculada ao se somar ou subtrair metade do valor da média daquela dimensão ao valor da instância corrente. Na Figura 48 é apresentado o processo descrito de sobreamostragem sobre um *singleton* com duas dimensões. Nesta figura há duas dimensões x e y, na qual x' é a média de todos os valores na dimensão x e y' é a média de todos os valores na dimensão y. O somar e o subtrair de x' e y' ocasionam a criação de quatro (ou $2d$) instâncias sintéticas.

Tabela 30 – Desempenho do VDBC.M2 obtido com as métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,7414±0,0173	15420±28,7889	-3,6201
Arrhythmia	0,7616±0,0374	561,4±28,5797	-0,5525
Balance Scale	0,6899±0,0265	545,8±21,5917	-0,0916
Car Evaluation	0,7345±0,0256	7949,4±110,1376	-4,7504
Contraceptive	0,5171±0,0036	15781±284,9714	-12,3919
Dermatology	0,8525±0,0571	1277,6±340,5493	-3,3634
E.coli	0,8836±0,0375	506,8±5,3572	-0,8854
Glass	0,8563±0,0273	385,4±10,1390	-1,2512
Hayes-Roth	0,6855±0,0456	193,6±5,6391	-0,5125
Horse	0,5972±0,0286	2334,4±30,3859	-6,9727
Nursery	0,8293±0,0222	9338,4±121,4096	0,0993
Page Blocks	0,8295±0,0089	26224±272,6329	-4,9894
Post Operative	0,6851±0,0308	140,2±6,3797	-0,9472
Soybean	0,9570±0,0127	864±14,2127	-0,7143
Wine	0,7801±0,0289	1046,8±286,9298	-6,3511
Yeast	0,7018±0,0364	2327±38,0197	-0,9601
Zoo	0,9191±0,0296	77,8±0,8367	0,0371

Após o processo de sobreamostragem o VDBC é executado normalmente. Os resultados, expressos como média e desvio-padrão da métrica MAUC e a quantidade de protótipos gerados, além da taxa de redução, são apresentados na Tabela 30. Note que a taxa de redução para quase todas as bases está negativa, o que significa que a quantidade de protótipos gerados foi maior que a quantidade de instâncias de treinamento.

Com uma rápida observação é possível perceber que o VDBC.M2 manteve o desempenho ou teve uma piora em relação ao VDBC, ao custo de um aumento no tamanho do conjunto de treinamento. Este aumento chegou a ser de mais de dez vezes a quantidade de instâncias, como no caso da base *Contraceptive*. Algumas bases não estão presentes na tabela devido ao tempo de processamento ter sido muito longo. Uma vez que não houve melhoria e praticamente não houve redução da quantidade de instâncias nas bases de dados, o VDBC.M2 demonstrou ser uma modificação insatisfatória.

Uma alternativa para esta estratégia é se ater aos parâmetros do SMOTE, ou seja, definir a proporção de aumento de uma classe, e definir também a quantidade de vizinhos observados para a criação de instâncias sintéticas. Novamente o cenário de múltiplas classes faz com que a escolha dos parâmetros seja mais complexa.

A.3 VDBC.M3

O tratamento sobre a sobreposição de bordas entre as classes motivou a terceira modificação do VDBC. Uma maneira de se calcular a sobreposição de bordas existente em uma base de dados é através da Taxa Generalizada de Fisher (F_{gen}) (MOLLINEDA; SÁNCHEZ; SOTOCA, 2005; MALDONADO; MONTECINOS, 2014). Esta taxa calcula o grau de separação existente entre as classes de uma base de dados, e é definida como:

$$F_{gen} = \frac{\sum_{k=1}^{C} n_k \delta(m, m_k)}{\sum_{k=1}^{C} \sum_{i=1}^{n_k} \delta(x_i^k, m_k)} \tag{A.1}$$

na qual n_k se refere ao número de instâncias da classe k, δ é a distância no espaço amostral, m é a média global, m_k é a média da classe k, e x_i^k é a instância i da classe k. A média global e a média de cada classe são, literalmente, o centroide de todas as instâncias e o centroide de cada classe, respectivamente.

Uma questão importante a ser destacada é que, como a taxa calcula o grau de separação entre as classes, logo, quanto maior o seu valor, mais separadas estão as classes. Por outro lado, quanto menor o valor, mais próximas estão as classes e, provavelmente, terão bordas sobrepostas.

As bordas sobrepostas podem ser consideradas *regiões difíceis*, ou complexas, do espaço amostral. Essas regiões consistem no espaço onde as fronteiras de duas ou mais classes se encontram, entretanto não são facilmente separáveis. A Figura 10, na primeira seção do Capítulo 3, apresenta um exemplo de uma base binária cujas bordas das classes são sobrepostas.

O VDBC.M3 procura os pares de classes cujo valor do F_{gen} seja menor ou igual a 0,15, valor definido empiricamente. Para cada par são encontrados os valores mínimos e máximos de cada atributo, além da quantidade de instâncias da maior classe do par. A partir de então, instâncias sintéticas são criadas dentro da faixa de valores encontrados. A classe de cada nova instância é selecionada aleatoriamente entre as classes do par. Após a criação de todas as novas instâncias sintéticas o VDBC é executado normalmente.

Na Tabela 31 são apresentados os valores de F_{gen} para todas as bases de dados. É interessante notar que existe uma correlação linear moderada (0,6123) entre os valores de F_{gen} e os resultados obtidos pelo VDBC. Em relação aos resultados do VDBC.M3, apresentados na Tabela 32, também há uma correlação linear positiva moderada (0,6299). Isto sugere que a distância entre as classes exerce influência sobre o algoritmo, de forma que quanto mais próximas são as classes, mais difícil é sua classificação.

Pode ser observado que verificar os pares de classes que são mais próximos, e criar instâncias sintéticas entre os mesmos aparentemente não melhora os resultados. As correlações lineares, como mostrado anteriormente, possuem valores próximos, e os desempenhos do VDBC e sua modificação corrente também são parecidos. A taxa de redução,

Tabela 31 – Taxa F_{gen} para cada base de dados.

Base	F_{gen}	**Base**	F_{gen}
Abalone	0,5975	Nursery	0,4171
Arrhythmia	0,3469	Page Blocks	0,3822
Balance Scale	0,4534	Post Operative	0,0587
Car Evaluation	0,2967	Satimage	1,4764
Contraceptive	0,2235	Shuttle	0,5625
Dermatology	0,4718	Soybean	1,2806
E.coli	1,4425	Thyroid	0,0352
Gene	0,1336	Wine	1,8162
Glass	0,7447	Yeast	0,5980
Hayes-Roth	0,3222	Zoo	1,3825
Horse	0,4683		

entretanto, está relacionada ao VDBC.M3. Enquanto no algoritmo original não há correlação entre a taxa de redução de instâncias e o F_{gen}, no VDBC.M3 há correlação. Quanto menor o valor do F_{gen}, ou quanto mais próximas são as classes, menor é a taxa de redução. Em outras palavras, quanto mais próximas as classes, mais protótipos foram gerados.

Apesar de ter havido real redução na quantidade de instâncias de treinamento, já que os protótipos substituem as instâncias, esta redução foi menor que no algoritmo original. Além disso, não houve melhoria no desempenho. Uma possível maneira de se contornar tais resultados é com a modificação de alguns parâmetros e estratégias. Por exemplo, considerar um valor de F_{gen} diferente para selecionar pares de classes; modificar a quantidade de instâncias sintéticas criadas; ou modificar a estratégia de atribuir uma classe à nova instância sintética. Os possíveis resultados com tais modificações ainda não foram objeto de estudo.

Tabela 32 – Desempenho do VDBC.M3 utilizando as métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8199±0,0109	2921,8±19,7914	0,1246
Arrhythmia:	0,7859±0,0259	274±7,6485	0,2423
Balance Scale	0,7013±0,0174	422,4±9,0995	0,1552
Car Evaluation	0,8334±0,0399	1277,4±14,8257	0,0760
Contraceptive	0,5741±0,0207	1121,2±11,5845	0,0485
Dermatology	0,9025±0,0502	196,8±3,8341	0,3279
E.coli	0,8867±0,0265	209,4±4,3359	0,2210
Gene	0,6777±0,0166	2256,2±24,2012	0,1159
Glass	0,8674±0,0669	163±3,6742	0,0479
Hayes-Roth	0,7528±0,0633	54,2±22,2194	0,5766
Horse	0,6612±0,0508	212,2±24,1806	0,2753
Nursery	0,8160±0,0124	5677,2±24,4888	0,4524
Page Blocks	0,7443±0,0426	5570,6±23,3838	-0,2723
Post Operative	0,6488±0,1174	76±1,8708	-0,0556
Satimage	0,9476±0,0041	2629,8±13,7004	0,4892
Shuttle	0,9332±0,0435	27915±54,0805	0,3984
Soybean	0,9643±0,0130	266,8±7,2595	0,4706
Thyroid	0,7653±0,0370	7398,2±33,3047	-0,2844
Wine	0,8015±0,0400	95,6±3,7815	0,3287
Yeast	0,6888±0,0642	1131,2±12,9306	0,0472
Zoo	0,9833±0,0160	27,8±2,5884	0,6559

A.4 VDBC.M4

Outra maneira simples de detectar bordas entre classes é utilizando a técnica Tomek Links (TOMEK, 1976). Um *Tomek Link* consiste em duas instâncias de classes diferentes as quais são o vizinho mais próximo uma da outra. Formalmente, dadas duas instâncias $x \in c_i$ e $y \in c_j$, na qual $i \neq j$ e $c = 1, ..., n$ é o conjunto de classes, $\delta(x, y) < \delta(x, m)$ e $\delta(x, y) < \delta(y, m)$, no qual m é qualquer instância de qualquer classe, além de x e y. Em outras palavras, a distância entre x e y é menor que a distância entre x e qualquer outra instância, e menor que a distância entre y e qualquer outra instância. Havendo um Tomek Link, supõe-se que as instâncias estejam nas bordas de suas respectivas classes.

No VDBC.M4, em vez da sobreamostragem, optou-se pela subamostragem a partir dos pares de instâncias que formam Tomek Links. Até então, a utilização de sobreamostragem não melhorou o desempenho do algoritmo, ao passo que a quantidade de instâncias pouco diminuiu, ou até mesmo aumentou.

A nova estratégia procura os pares de instâncias que são Tomek Links. Para cada par

Tabela 33 – Desempenho do VDBC.M4 utilizando as métricas MAUC, Geração e Redução de Instâncias

Nome	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8168±0,0104	2236,6±7,3348	0,3299
Arrhythmia	0,7592±0,0528	206,8±4,9699	0,4281
Balance Scale	0,6982±0,0072	233,8±5,2631	0,5324
Car Evaluation	0,8634±0,0375	485,8±17,0939	0,6486
Contraceptive	0,5984±0,0361	667,6±17,0968	0,4335
Dermatology	0,9111±0,0216	128,4±3,2094	0,5615
E.coli	0,8833±0,0469	136,8±4,0866	0,4911
Gene	0,6944±0,0194	1244,8±32,9348	0,5122
Glass	0,8890±0,0147	93,4±5,0299	0,4544
Hayes-Roth	0,8074±0,0594	52±1,8708	0,5937
Horse	0,6800±0,0083	158,8±3,5637	0,4577
Nursery	0,8362±0,0342	2964,4±28,7889	0,7141
Page Blocks	0,7662±0,0183	2057,8±9,0940	0,5300
Post Operative	0,6297±0,0541	34,8±2,5884	0,5167
Satimage	0,9469±0,0087	2091,8±19,1494	0,5937
Shuttle	0,9408±0,0635	20756±24,9660	0,5527
Soybean	0,9679±0,0134	231±6,7823	0,5417
Thyroid	0,7643±0,0190	2810,6±16,3799	0,5120
Wine	0,8431±0,0469	74±3,5355	0,4803
Yeast	0,7000±0,0283	668,6±7,3348	0,4368
Zoo	0,9810±0,0181	27,2±2,2804	0,6634

encontrado, a instância pertencente à maior classe é excluída do conjunto de treinamento. Entretanto, se o tamanho da maior classe for menor ou igual à quantidade de *folds*, não há exclusão da instância. Uma vez que os testes foram executados com cinco *folds*, toda classe com cinco ou menos instâncias de treinamento não teve instâncias removidas.

Outra modificação importante está na verificação das instâncias. Até então, o algoritmo verificava sequencialmente cada instância de treinamento, com reposição. A partir do VDBC.M4 as instâncias são selecionadas aleatoriamente, e após a verificação são excluídas do conjunto de treinamento. Além das modificações citadas, o algoritmo segue sua estrutura tradicional.

A verificação aleatória impede que alguma classe seja beneficiada caso uma sequência de suas instâncias sejam as primeiras da fila. Se tal classe possa ser considerada majoritária, tal benefício deve ser impedido em aprendizado sobre classes desbalanceadas. Ainda, pode acontecer das menores classes serem as últimas a serem verificadas e, portanto, serem prejudicadas de alguma forma. A exclusão das instâncias já verificadas do conjunto de treinamento tem por objetivo uma redução de custo computacional.

Os resultados obtidos pelo VDBC.M4 são apresentados na Tabela 33. Uma comparação rápida entre a modificação corrente e o VDBC mostra que foi possível manter o desempenho com uma menor quantidade de protótipos. Ainda, é possível que tenha havido melhora de desempenho em algumas bases. As comparações estatísticas presentes na seção 3.2.1 tratam tais observações com mais detalhes.

A.5 VDBC.M5

Enquanto as modificações anteriores focavam em um processamento do conjunto de treinamento antes da geração de protótipos, o VDBC.M5 modifica a própria estratégia de geração de protótipos. Uma vez que a estrutura do algoritmo foi modificado, o VDBC.M5 pode também ser considerado uma segunda versão do VDBC.

Algoritmo 5: Pseudo-código do VDBC.M5

```
1: /*Dadas todas as instâncias de treinamento*/
2: Encontre gRaio, i.e., a metade da menor distância entre os pares de instâncias;
3: Transforme todas as instâncias em protótipos, com raio = 0;
4: Organize todas as instâncias da menor para a maior classe;
5: enquanto ∃ instância cuja região possa crescer faça
6:    para cada classe, da menor para a maior faça
7:       para instância aleatória da classe corrente faça
8:          se região da instância pode crescer então
9:             Calcule nRraio = raio + gRaio;
10:            se nRaio tocar ou traspassar a região de outra instância então
11:               se as instâncias forem da mesma classe então
12:                  Crie um centroide entre as mesmas, onde raio = max(raio_1, raio_2);
13:               senão
14:                  Configure a instância como impedida de crescer;
15:               fim do se
16:            fim do se
17:         fim do se
18:      fim do para
19:   fim do para
20: fim do enquanto
21: /*Dadas todas as instâncias de teste*/
22: para cada instância faça
23:    Classifique de acordo com o centroide mais próximo;
24: fim do para
```

O pseudo-algoritmo do VDBC.M5 pode ser visto no Algoritmo 5. Esta modificação transforma as instâncias em centroides de hiperesferas. À medida que os raios das hiperesferas vão crescendo, as mesmas são forçadas a se juntar a outras, caso sejam da mesma classe, ou pararem de crescer, se forem de classes diferentes.

O primeiro passo é encontrar a menor distância entre quaisquer pares de instâncias. A metade dessa distância é o valor de crescimento do raio das hiperesferas. O passo seguinte

Tabela 34 – Desempenho do VDBC.M5 utilizando as métricas MAUC, Geração e Redução de Instâncias

Nome	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8170±0,0223	2821,8±16,1462	0,1545
Arrhythmia	0,8006±0,0321	183,4±10,9909	0,4928
Balance Scale	0,7032±0,0515	129,2±7,5631	0,7416
Car Evaluation	0,8883±0,0367	118,6±30,5663	0,9142
Contraceptive	0,5824±0,0294	773,8±11,1669	0,3433
Dermatology	0,8687±0,0571	61,8±5,2631	0,7889
E.coli	0,8453±0,0690	66,8±7,9498	0,7515
Gene	0,6529±0,0250	1186,2±24,0250	0,5352
Glass	0,8419±0,0419	65,4±4,2190	0,6180
Hayes-Roth	0,7239±0,0160	68,2±7,1903	0,4672
Horse	0,6478±0,0611	123±4,6904	0,5799
Nursery	0,8476±0,0345	688,4±42,4064	0,9336
Post Operative	0,6056±0,0567	41±5,6569	0,4306
Satimage	0,9017±0,0106	576±23,2809	0,8881
Shuttle	0,8800±0,0673	151,4±5,2726	0,9967
Wine	0,7561±0,0655	51,4±2,0736	0,6390
Zoo	0,9905±0,0130	11,6±1,3416	0,8564

é transformar todas as instâncias de treinamento em protótipos. Isto significa que além dos atributos e classe, cada instância terá um valor real para o raio de sua hiperesfera e um valor *booleano* indicando se pode ou não aumentar seu raio. De início, todos os protótipos possuem o valor do raio igual a 0 e o crescimento do raio como *true*.

O terceiro passo é ordenar as classes, de forma ascendente em relação à quantidade de instâncias, a fim de organizar o crescimento das hiperesferas. Como as bases são desbalanceadas, o crescimento acontece primeiro nas menores classes. É mais comum que as maiores classes sejam mais *coesas*, i.e., tenham uma concentração maior de instâncias e suas regiões sejam mais facilmente delineadas. Quanto menor é uma classe, maior a possibilidade dessa *coesão* não existir, e de ser mais difícil a delimitação de sua região. Portanto, é interessante permitir que as hiperesferas das menores classes tenham preferência em seu crescimento, pois assim é menor a chance de sua generalização ser prejudicada por uma classe maior.

Após a ordenação das classes pelo seu tamanho, as instâncias de cada classe são selecionadas aleatoriamente. Caso estejam *aptas* a crescerem, haverá uma soma do seu raio com o valor de crescimento de raio. Após essa soma, é feita uma verificação. Caso a hiperesfera corrente, ao crescer, toque ou traspasse a região de outra hiperesfera, uma de duas medidas será tomada: (1) junção das duas hiperesferas, caso sejam da mesma classe

ou (2) as mesmas são configuradas para pararem de crescer. O processo é repetido até que não haja qualquer hiperesfera que possa aumentar seu raio.

Na Tabela 34 são apresentados os resultados obtidos pela modificação corrente. Quatro bases de dados (*Page Blocks*, *Soybean*, *Thyroid* e *Yeast*) estão ausentes pelo fato de o valor de crescimento do raio ser pequeno. Isto fez com que a cada iteração o crescimento das hiperesferas fosse praticamente nulo, tendo por consequência um gasto de tempo impraticável para o experimento ou sua utilização prática. Contudo, nas bases presentes na tabela, é possível perceber que houve grande redução na quantidade de protótipos gerados.

APÊNDICE B – DETALHES DAS MODIFICAÇÕES DO DCIA

Neste apêndice todas as modificações realizadas sobre o DCIA são apresentadas em detalhes. Cada seção consiste na apresentação de uma das modificações, juntamente aos resultados obtidos e suas respectivas análises.

B.1 DCIAGSA.M1

A transformação ou normalização dos dados é uma ferramenta de pré-processamento comumente usada em mineração de dados (HAN; KAMBER; PEI, 2012; JAIN; BHANDARE, 2011). A normalização dos dados procura oferecer a todos os atributos um peso igual, escalando os valores dentro de uma faixa de intervalo, e.g., [0, 1]. Desta forma, a padronização é particularmente útil para os algoritmos de classificação, inclusive os que se utilizam de medidas de distância como o *kNN* (HAN; KAMBER; PEI, 2012).

Duas das mais conhecidas técnicas de normalização de dados são a *min-max* e a *z-score* ou *standard score* (HAN; KAMBER; PEI, 2012; JAIN; BHANDARE, 2011). A primeira realiza uma transformação linear nos dados. Supondo que min_A e max_A são, respectivamente, os valores mínimo e máximo de um atributo A, todos os valores são transformados da seguinte forma:

$$A'_i = \frac{A_i - min_A}{max_A - min_A} \tag{B.1}$$

no qual A'_i será o novo valor de A_i. Na Equação B.1 todos os valores de A serão reescalados na faixa [0, 1]. Contudo se outra faixa de valores for desejada, a equação pode ser adaptada para:

$$A'_i = \frac{A_i - min_A}{max_A - min_A}(b - a) + a \tag{B.2}$$

no qual a e b são os novos valores mínimo e máximo, respectivamente.

A segunda técnica de normalização dos dados se baseia na média e desvio-padrão dos valores de um atributo A. O $z - score$ normaliza os valores da seguinte forma:

$$A'_i = \frac{A_i - \bar{A}}{\sigma_A} \tag{B.3}$$

no qual $\bar{A}$ é a média aritmética dos valores de A e σ_A é o desvio-padrão.

A normalização com o $z - score$ é útil quando não há informações sobre os valores mínimo e máximo de algum atributo, ou quando há *outliers* que dominem a normalização *min-max* (HAN; KAMBER; PEI, 2012). Na pesquisa realizada os dados são comumente divididos em dois conjuntos. Desta forma, não há garantias que os valores extremos para

Tabela 35 – Desempenho do DCIAGSA.M1 utilizando as métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8247±0,0082	284,8±11,1221	0,9147
Arrhythmia	0,7634±0,0632	70,2±9,2574	0,8059
Balance Scale	0,7983±0,0628	7,6±2,7019	0,9848
Car Evaluation	0,9126±0,0260	37,8±5,8052	0,9727
Contraceptive	0,6122±0,0187	17,6±2,5100	0,9851
Dermatology	0,9672±0,0305	10,6±0,8944	0,9638
E.coli	0,9035±0,0466	32,2±2,5884	0,8802
Gene	0,7457±0,0190	50,2±6,3797	0,9803
Glass	0,8745±0,0638	29,2±6,2209	0,8294
Hayes-Roth	0,8044±0,0435	15,2±7,9498	0,8813
Horse	0,7069±0,0449	24,6±12,0540	0,9160
Nursery	0,8413±0,0414	13±1,8708	0,9987
Page Blocks	0,8702±0,0286	52,4±10,7842	0,9880
Post Operative	0,6417±0,0636	25±7,2801	0,6528
Satimage	0,9234±0,0059	65,2±4,1473	0,9873
Shuttle	0,9800±0,0059	55,2±9,9599	0,9988
Soybean	0,9722±0,0080	41,8±4,3243	0,9171
Thyroid	0,8752±0,0151	22±5,1478	0,9962
Wine	0,9844±0,0117	5,4±1,1402	0,9621
Yeast	0,8297±0,0349	83,2±19,7408	0,9299
Zoo	0,9863±0,0188	8,6±1,1402	0,8936

cada atributo sejam conhecidos na normalização. Ainda, *outliers* podem exercer influência significativa em bases de dados desbalanceadas (MALDONADO; MONTECINOS, 2014; HAIXIANG et al., 2017). Estas razões, aliadas ao fato de que o DCIAGSA teve piora de desempenho em duas bases que possuem uma ampla faixa de valores em pelo menos um atributo, fez com que o *z-score* fosse escolhido para normalizar os dados.

A normalização foi implementada da seguinte forma. Após a divisão dos dados, todas as instâncias do conjunto de treinamento foram normalizadas de acordo com a Equação (B.3). O conjunto de teste utilizou a mesma equação, entretanto, a média e o desvio-padrão utilizados foram os do conjunto de treinamento. A normalização do conjunto de teste foi feito levando em conta o conjunto de treinamento porque as instâncias de teste devem ser parte de um conjunto *desconhecido*. Contudo, é necessário supor que as características do conjunto de teste sejam similares às do conjunto de treinamento. Portanto, o conjunto de teste é normalizado se adequando ao conjunto de treinamento.

Após a normalização dos dados, o DCIAGSA é executado normalmente. Na Tabela 35 são apresentados os resultados obtidos pelo DCIAGSA com sua primeira modificação.

A partir de uma comparação rápida é possível notar que nas bases *Dermatology*, *Nursery*, *Page Blocks*, *Thyroid* e *Wine* houve um *salto* positivo no desempenho. Todas essas bases possuem notável variedade nos valores de seus atributos, classes muito próximas umas das outras, ou ambos os fatores ao mesmo tempo. Junto a isso, a média de protótipos gerados diminuiu para 0,0744, ou seja, em média o conjunto de treinamento está sendo reduzido a 7% do seu tamanho.

Quanto à base *Satimage*, o classificador manteve um desempenho similar, apesar das diferenças de valores dos atributos. A razão encontrada para tal acontecimento se dá na junção das características da base, i.e., pouco desbalanceamento, classes separadas, e uma grande quantidade de instâncias. Um indicativo de que suas características influenciam na classificação está nos valores de desvio-padrão das médias de desempenho em todos os experimentos, sempre muito baixos. Portanto, apesar de ter havido uma diminuição de 2 pontos percentuais no desempenho, a classificação ainda pode ser considerada muito boa, uma vez que está acima dos 90%. Outro fator que conta favoravelmente ao DCIAGSA.M1 é que o desempenho próximo ao do VDBC.M4 e similar ao do DCIAGSA foi atingido com uma quantidade menor de protótipos gerados.

Em relação à modificação implementada, é interessante notar que a simples adição de normalização nos dados pode resultar em melhoria de desempenho. Contudo, ainda assim, algumas bases são complexas o suficiente para que o classificador não seja capaz de obter uma taxa de classificação superior a 70%. Por outro lado, bases como *Shuttle*, com menos de 1% de instâncias de treinamento possuem informação suficiente para que o classificador *1NN* consiga ter um sucesso próximo de 100%.

B.2 DCIAGSA.M2

O *dataset shift* é um dos fatores que pode afetar a classificação tanto em bases balanceadas quanto em bases desbalanceadas. Nas bases desbalanceadas o efeito pode ser mais severo (LÓPEZ; FERNÁNDEZ; HERRERA, 2014). Este fator consiste em um conjunto de dados ser representado com distribuições diferentes quando divididos os conjuntos de treinamento e teste (LÓPEZ et al., 2013; MORENO-TORRES; SÁEZ; HERRERA, 2012). Isso pode acontecer devido à natureza intrínseca do problema ou pode ser introduzido através de algum esquema de *cross-validation* (MORENO-TORRES; SÁEZ; HERRERA, 2012).

Dois dos tipos de *dataset shift* mais estudados são *prior probability shift* e *covariate shift* (MORENO-TORRES; SÁEZ; HERRERA, 2012; LÓPEZ; FERNÁNDEZ; HERRERA, 2014). O primeiro consiste em uma diferença de distribuição das classes e pode ser contornado com um *cross-validation* estratificado. O segundo consiste em diferentes distribuições dos valores dos atributos.

O primeiro tipo de *dataset shift* já é tratado nos testes que foram conduzidos até então. Entretanto, é possível que ao se tratar, ou prevenir o segundo tipo, o desempenho de classificação possa melhorar, como aconteceu quando os dados foram normalizados. Ló-

Tabela 36 – Desempenho do DCIAGSA.M2 utilizando as métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8287±0,0204	247±44,1984	0,9260
Arrhythmia	0,7728±0,0512	72,6±7,0214	0,7992
Balance Scale	0,7988±0,0413	6,8±2,5884	0,9864
Car Evaluation	0,9256±0,0262	36,4±5,4129	0,9737
Contraceptive	0,6287±0,0275	16,8±6,4576	0,9857
Dermatology	0,9731±0,0149	10,8±1,6432	0,9631
E.coli	0,9180±0,0354	30,6±4,6152	0,8862
Gene	0,7470±0,0105	49,6±6,9498	0,9806
Glass	0,8890±0,0373	29,6±6,6558	0,8271
Hayes-Roth	0,8034±0,0774	21,4±9,1269	0,8328
Horse	0,6883±0,0372	22,2±13,6638	0,9242
Nursery	0,8284±0,0269	12,4±4,2778	0,9988
Page Blocks	0,8707±0,0404	43±4,8477	0,9902
Post Operative	0,6617±0,1055	24,8±12,6570	0,6556
Satimage	0,9272±0,0091	79±12,0623	0,9847
Shuttle	0,9769±0,0112	54,8±12,1943	0,9988
Soybean	0,9717±0,0085	39,8±3,0332	0,9210
Thyroid	0,8806±0,0127	18,8±4,2661	0,9967
Wine	0,9819±0,0189	6±1,5811	0,9579
Yeast	0,8343±0,0151	94,6±10,1637	0,9203
Zoo	0,9862±0,0189	7,8±1,3038	0,9035

pez, Fernández e Herrera (2014) concluíram que uma maneira adequada de se abordar o segundo tipo de *dataset shift* em bases desbalanceadas é utilizando o DOB-SCV (*Distribution Optimally Balanced Stratified Cross-Validation*) (MORENO-TORRES; SÁEZ; HERRERA, 2012). O estudo contudo foi realizado apenas com bases desbalanceadas binárias. Neste trabalho o DOB-SCV é testado em bases com múltiplas classes desbalanceadas.

O DOB-SCV é uma modificação do DB-SCV (*Distribution Balanced Stratified Cross-Validation*) (ZENG; MARTINEZ, 2000; MORENO-TORRES; SÁEZ; HERRERA, 2012). Ambos os métodos adicionam uma consideração extra à estratégia de partição de dados na tentativa de reduzir o *covariate shift* ao mesmo tempo em que previne o *prior probability shift*. Seu funcionamento para um particionamento em $k-folds$ é da seguinte forma. Uma instância aleatória é selecionada, e junto a ela seus $k-1$ vizinhos mais próximos da mesma classe que não estejam associados a qualquer *fold*. A primeira instância é associada ao primeiro *fold* e seus vizinhos são associados aos demais *folds*. O processo é repetido até que todas as instâncias estejam associadas a alguma partição.

O DCIAGSA.M2 implementa o DOB-SCV como o método de partição dos dados. O

algoritmo passa a ter os seguintes passos gerais:

1. Particionar a base de dados em *k-folds* com DOB-SCV;
2. Normalizar os dados de treinamento e teste;
3. Executar DCIAGSA.

Na Tabela 36 são apresentados os resultados obtidos com a modificação corrente. Aparentemente não houve alterações nos resultados em relação à modificação anterior. É importante notar que a execução dos experimentos com *5-fold* pode não ter sido suficiente para evidenciar alguma modificação (KOHAVI, 1995; LÓPEZ; FERNÁNDEZ; HERRERA, 2014).

B.3 DCIASGSA.M1 E DCIASGSA.M2

As primeiras duas modificações do DCIASGSA envolvem uma hibridização de sobreamostragem e subamostragem no conjunto de treinamento. As estratégias utilizadas nessas duas modificações já foram utilizadas no VDBC, nas modificações 2 e 4, respectivamente. Entretanto, até então, tais estratégias foram utilizadas separadamente.

O DCIASGSA.M1 utiliza Tomek Links para subamostragem dos dados, seguido de uma sobreamostragem inspirada no método SMOTE. O DCIASGSA.M2 utiliza as mesmas estratégias, contudo na ordem inversa, i.e., primeiro realiza a sobreamostragem para, em seguida, realizar uma subamostragem.

A subamostragem acontece da seguinte forma. Todas as instâncias de treinamento que formam um Tomek Link são selecionadas e inseridas em um vetor. O vetor é percorrido, instância por instância, sendo realizada a verificação de classe. Caso a classe da instância corrente seja maior que o número de *folds*, a instância será apagada seguindo uma condição. A condição é um número aleatório em uma distribuição uniforme entre 0 e 1 ser menor ou igual à probabilidade *a priori* da classe. Desta forma, as maiores classes terão menor probabilidade de serem muito afetadas pela redução de suas instâncias, ao passo em que somente as classes que *dominam* a base, ou seja, as que são muito grandes, é que terão maior probabilidade de serem reduzidas.

Quanto à sobreamostragem, a diferença em relação à estratégia utilizada no VDBC está sobre a seleção das classes minoritárias. A partir de então, as classes selecionadas serão aquelas que possuem até $2 * folds$ instâncias de treinamento. Comumente, a quantidade de *folds* é 5 ou 10, portanto, esta estratégia seleciona as classes que possuem até 10 ou 20 instâncias de treinamento. O processo de criação de instâncias sintéticas se mantém, ou seja, para cada par de instâncias de uma classe, uma instância sintética é criada entre as mesmas. Se houver uma única instância de treinamento, o processo apresentado no Apêndice A.2 e Figura 48 é utilizado.

Tabela 37 – Desempenho do DCIASGSA.M1 utilizando as métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8193±0,0130	265,8±39,7077	0,9204
Arrhythmia	0,7852±0,0339	72,6±8,3247	0,7992
Balance Scale	0,8502±0,0348	4,8±1,3038	0,9904
Car Evaluation	0,9251±0,0200	29,6±8,0498	0,9786
Contraceptive	0,6253±0,0119	18±3,3912	0,9847
Dermatology	0,9736±0,0220	11,2±1,9235	0,9617
E.coli	0,9108±0,0328	29±6,0415	0,8921
Gene	0,7460±0,0094	45,2±5,3104	0,9823
Glass	0,8489±0,0623	23±3,6742	0,8657
Hayes-Roth	0,7448±0,0825	14,6±6,6182	0,8859
Horse	0,7000±0,0455	20,8±8,1056	0,9290
Nursery	0,8167±0,0225	13,2±2,1679	0,9987
Page Blocks	0,8687±0,0199	34,8±11,4543	0,9921
Post Operative	0,6306±0,1290	22±9,1652	0,6944
Satimage	0,9197±0,0033	70,6±15,1096	0,9863
Shuttle	0,9716±0,0143	44,8±13,5536	0,9990
Soybean	0,9717±0,0122	40,8±4,0866	0,9190
Thyroid	0,8780±0,0027	23±2,9155	0,9960
Wine	0,9789±0,0071	6,4±1,1402	0,9551
Yeast	0,8381±0,0504	82,6±14,9599	0,9304
Zoo	0,9704±0,0211	7,4±0,5477	0,9084

Nas Tabelas 37 e 38 são apresentados os resultados obtidos pelas duas modificações do DCIASGSA. As duas modificações tiveram seus resultados considerados estatisticamente equivalentes ao DCIASGSA. Levando-se em conta o desempenho, os testes estatísticos revelam que as modificações são equivalentes ao algoritmo original em todas as bases. Quanto aos protótipos gerados, as duas modificações diminuíram a quantidade na base *Glass*, de 31 protótipos para cerca de 23, em média. No restante das bases a quantidade de protótipos gerados é estatisticamente equivalente.

Tais modificações, apesar de terem resultado em melhoras no VDBC, não foram suficientes para melhorar o desempenho do DCIASGSA. A modificação do conjunto de treinamento anterior à geração de protótipos, com diminuição das maiores classes e aumento das menores classes, somente teve efeito em uma única base. Portanto, uma vez que houve um aumento de custo computacional durante o pré-processamento, não é recomendável a utilização dessas modificações.

Tabela 38 – Desempenho do DCIASGSA.M2 utilizando das métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8242±0,0084	290,2±28,7785	0,9131
Arrhythmia	0,7700±0,0303	69±11,5974	0,8092
Balance Scale	0,8086±0,0223	4±0,7071	0,9920
Car Evaluation	0,9145±0,0254	33,8±3,4205	0,9755
Contraceptive	0,6231±0,0130	16,6±1,1402	0,9859
Dermatology	0,9802±0,0074	10,8±2,1679	0,9631
E.coli	0,9180±0,0270	34±4,7958	0,8735
Gene	0,7488±0,0105	48,6±4,7749	0,9810
Glass	0,8876±0,0315	23,8±5,1186	0,8610
Hayes-Roth	0,7707±0,0662	11,8±3,6332	0,9078
Horse	0,6970±0,0285	18±9,8234	0,9385
Nursery	0,8228±0,0341	11,2±2,8636	0,9989
Page Blocks	0,8840±0,0130	48,6±11,6103	0,9889
Post Operative	0,6258±0,0777	22,6±4,5056	0,6861
Satimage	0,9289±0,0060	76,2±12,2963	0,9852
Shuttle	0,9687±0,0272	49±14,7139	0,9989
Soybean	0,9736±0,0115	39,4±3,8471	0,9218
Thyroid	0,8848±0,0208	18,8±5,9330	0,9967
Wine	0,9878±0,0092	5,2±0,4472	0,9635
Yeast	0,8272±0,0236	86,4±13,0499	0,9272
Zoo	0,9862±0,0147	8,2±0,4472	0,8985

B.4 DCIASGSA.M3 E DCIASGSA.M4

A seleção de *features* é um processo de identificação e seleção das características, ou atributos, mais importantes de uma base de dados. Ao se utilizar somente tais atributos selecionados espera-se a melhora do desempenho dos classificadores (MALLENAHALLI; SARMA, 2018; EMARY; ZAWBAA; HASSANIEN, 2016; GU; CHENG; JIN, 2018). Além disso, a seleção de atributos é um tópico recente no estudo de problemas em bases de dados desbalanceadas (MOAYEDIKIA et al., 2017).

Para fins de avaliação dois algoritmos de redução de dimensionalidade foram utilizados: *t-distributed Stochastic Neighbor Embedding* (t-SNE) (MAATEN; HINTON, 2008) e *Feature Selection for Classification Using Neighborhood Component Analysis* (FSCNCA) baseado no algoritmo apresentado em (YANG; WANG; ZUO, 2012).

O t-SNE tem por objetivo ajudar na visualização de dados com alta dimensionalidade (MAATEN; HINTON, 2008). O mesmo é capaz de capturar de forma eficiente as características de estruturas locais ao mesmo tempo em que revela estruturas globais dos dados

Tabela 39 – Desempenho do DCIASGSA.M3 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,7046±0,0520	280,8±41,5656	0,9159
Arrhythmia	0,7027±0,0376	55,8±5,6745	0,8457
Balance Scale	0,8034±0,0626	5,8±1,6432	0,9884
Car Evaluation	0,6496±0,0378	27±11,4455	0,9805
Contraceptive	0,5708±0,0187	14,8±7,1204	0,9874
Dermatology	0,6034±0,1158	9,4±2,3022	0,9679
E.coli	0,6630±0,1243	25,8±4,9699	0,9040
Gene	0,5665±0,0407	19,2±5,3572	0,9925
Glass	0,6536±0,0906	28,2±11,8195	0,8353
Hayes-Roth	0,7890±0,0353	10,8±1,9235	0,9156
Horse	0,6135±0,0712	16,6±3,2094	0,9433
Nursery	0,6135±0,0235	35,8±7,8867	0,9965
Page Blocks	0,6560±0,0269	24,8±6,7602	0,9943
Post Operative	0,5931±0,0612	15,8±3,2711	0,7806
Satimage	0,7860±0,0683	25,2±4,7645	0,9951
Shuttle	0,6663±0,0458	84,2±9,1214	0,9982
Soybean	0,7966±0,0387	51±3,8079	0,8988
Thyroid	0,5926±0,0544	32,6±9,6592	0,9943
Wine	0,6927±0,1764	5,2±0,4472	0,9635
Yeast	0,6877±0,0525	62±11,5109	0,9478
Zoo	0,8299±0,0441	10,4±1,3416	0,8713

analisados, como a presença de *clusters* em diversas escalas (MAATEN; HINTON, 2008). Sua implementação no Matlab permite a escolha da dimensionalidade final dos dados. Neste trabalho, foi selecionado empiricamente o valor de quatro atributos, para fins de teste. Desta forma, após o processamento feito pelo t-SNE os dados terão apenas quatro atributos, ou dimensões.

O algoritmo FSCNCA é baseado no algoritmo NFSC (*Neighborhood Component Feature Selection*) (YANG; WANG; ZUO, 2012). Os detalhes de seu funcionamento podem ser consultados online, na página <https://www.mathworks.com/help/stats/neighborhood-component-analysis.html>.

Na Tabela 5 da Seção 3.2 é apresentada a quantidade de atributos presentes em cada base de dados. A base com maior quantidade de atributos é *Arrhythmia*, com 260. As bases *Balance Scale* e *Hayes-Roth* possuem a menor quantidade de atributos. Em resumo, as 21 bases possuem, em média, 28 atributos. Com essas duas modificações o objetivo é verificar se acrescentar o passo de seleção de atributos resulta em alguma melhoria ao algoritmo já existente.

Tabela 40 – Desempenho do DCIASGSA.M4 utilizando das métricas MAUC, Geração e Redução de Instâncias

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8300±0,0138	249±41,0061	0,9254
Arrhythmia	0,8116±0,0257	70±8,1548	0,8064
Balance Scale	0,8405±0,0191	4,8±1,4832	0,9904
Car Evaluation	0,9229±0,0289	38,2±1,3038	0,9724
Contraceptive	0,6590±0,0065	22,4±7,3007	0,9810
Dermatology	0,9631±0,0194	11,2±2,1679	0,9617
E.coli	0,9155±0,0214	30,4±9,8387	0,8869
Gene	0,7650±0,0190	49±3,8730	0,9808
Glass	0,8529±0,0496	24,6±9,8387	0,8563
Hayes-Roth	0,8065±0,0477	9,4±2,1909	0,9266
Horse	0,7123±0,0376	20,6±9,5812	0,9296
Nursery	0,8209±0,0393	10,4±2,5100	0,9990
Page Blocks	0,8791±0,0191	43,6±7,8294	0,9900
Post Operative	0,6137±0,0689	6,4±1,8166	0,9111
Satimage	0,9245±0,0093	83,4±15,5016	0,9838
Shuttle	0,9831±0,0075	59,2±5,3572	0,9987
Soybean	0,9736±0,0125	37±7,2111	0,9266
Thyroid	0,9013±0,0151	21,6±10,0648	0,9962
Wine	0,9802±0,0066	5,2±1,3038	0,9635
Yeast	0,8447±0,0243	73±5,3852	0,9385
Zoo	0,9901±0,0104	8,6±0,5477	0,8936

O DCIASGSA.M3 utiliza o t-SNE após a normalização dos dados. A redução de dimensionalidade é efetuado nos conjuntos de treinamento e teste de forma separada. Levando-se em conta que os conjuntos possuem características similares, espera-se que a redução para 4 dimensões em ambos os conjuntos seja coerente. O DCIASGSA.M4 utiliza o algoritmo FSCNCA no conjunto de treinamento, também após a normalização dos dados. O conjunto de teste também sofre redução de atributos, os quais correspondem aos atributos selecionados durante a fase de treinamento.

Nas Tabelas 39 e 40 são apresentadas as médias e desvios-padrões dos resultados obtidos por essas duas modificações. Os resultados apresentados são os das métricas MAUC e quantidade de protótipos gerados, além da taxa de redução do conjunto de treinamento. Ambas as modificações foram comparadas com o DCIASGSA.

Os testes estatísticos confirmam a inferioridade do DCIASGSA.M3, cujo baixo desempenho em praticamente todas as bases é evidente. A redução de dimensionalidade para uma quantidade fixa de atributos não se mostrou uma boa estratégia, apesar de ter conseguido, em alguns casos, uma quantidade menor de protótipos gerados. Contudo não

houve qualquer proveito da geração de um conjunto de treinamento menor, uma vez que houve piora de desempenho.

A estratégia adotada no DCIASGSA.M4 produziu um melhor desempenho, em relação ao DCIASGSA.M3. A redução de dimensionalidade, neste caso, buscou um subconjunto de atributos que resultasse em melhor desempenho. A classificação das bases normalizadas com um subconjunto de atributos manteve o mesmo desempenho e a mesma quantidade de protótipos gerados na maioria das bases. Contudo, na base *Post Operative* houve uma grande redução da quantidade de protótipos gerados, de cerca de 26 para cerca de 6 instâncias. Ao mesmo tempo, nas base *Zoo* houve melhora de desempenho.

Os dois pontos positivos encontrados com os testes do DCIASGSA.M4 mostram que há possibilidade de melhora de desempenho e diminuição da quantidade de protótipos gerados. Contudo, apenas dois algoritmos de redução de dimensões foram utilizados até então. Possivelmente outro algoritmo consiga produzir um subconjunto de atributos melhor para o modelo de classificação.

B.5 DCIAPSO.M1

Até então a métrica MAUC tem sido usada durante todos os processos do DCIA. Entretanto existem outras métricas adequadas para o contexto de múltiplas classes desbalanceadas. Uma delas é a métrica AUCarea. Esta métrica é mais sensível aos erros de AUCs de pares de classes, logo valores ruins terão maior efeito sobre o valor final do AUCarea (YIJING et al., 2016).

A utilização desta métrica, por ser mais sensível, possivelmente pode pressionar o ajuste dos protótipos para uma melhor configuração. Portanto, durante a execução do DCIAPSO, o ajuste das posições feita pelo PSO pode retornar resultados melhores. É importante notar que o cálculo do desempenho final é separado do cálculo do desempenho durante o ajuste de posições.

Na Tabela 41 é apresentado o desempenho final do DCIAPSO.M1. É evidente que a suposição formulada para esta modificação se provou falha. Aparentemente, não houve melhoras no desempenho e somente na base *Zoo* não houve grande aumento da quantidade de protótipos gerados.

A modificação foi comparada estatisticamente ao DCIAPSO. As comparações mostram que o desempenho do DCIAPSO.M1 é similar ao DCIAPSO em todas as bases, exceto na base *Yeast*, na qual a modificação obteve desempenho pior.

Se considerarmos que a métrica AUCarea é mais sensível, a expectativa era que ao encontrar regiões que retornem melhores valores, o desempenho final medido com MAUC fosse, também, melhor. Contudo, o cálculo do *fitness* com AUCarea aparentemente não permitiu tal convergência no PSO. Possivelmente, uma quantidade maior de iterações permita ao algoritmo convergir para regiões melhores. Isto, entretanto, se mostra como

Tabela 41 – Desempenho do DCIAPSO.M1 utilizando das métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8261±0,0086	350,6±33,5306	0,8950
Arrhythmia	0,7634±0,0345	121±9,0277	0,6654
Balance Scale	0,7198±0,0521	74±10,8858	0,8520
Car Evaluation	0,9390±0,0311	63,2±4,4385	0,9543
Contraceptive	0,6131±0,0206	69,8±6,7602	0,9408
Dermatology	0,9761±0,0112	59,2±1,0954	0,7978
E.coli	0,9061±0,0318	85,4±6,5803	0,6823
Gene	0,7539±0,0127	81,2±3,6332	0,9682
Glass	0,8813±0,0227	89,8±5,9330	0,4755
Hayes-Roth	0,6945±0,0844	74,8±3,7683	0,4156
Horse	0,6996±0,0527	72,6±6,6933	0,7520
Nursery	0,8245±0,0417	62,2±3,1937	0,9940
Page Blocks	0,8677±0,0387	87,8±9,1488	0,9799
Post Operative	0,5867±0,0530	66,2±6,6858	0,0806
Satimage	0,9256±0,0069	133±5,0990	0,9742
Shuttle	0,9605±0,0366	92,4±8,7920	0,9980
Soybean	0,9713±0,0102	90,8±4,5497	0,8198
Thyroid	0,8811±0,0089	64±1,8708	0,9889
Wine	0,9837±0,0134	29,4±13,6125	0,7935
Yeast	0,7573±0,0110	164±22,7046	0,8619
Zoo	0,9648±0,0395	12±10,0995	0,8515

uma desvantagem, uma vez que a utilização do MAUC como métrica do *fitness* permite a convergência para regiões similares em menos iterações.

O leitor pode ter ficado curioso quanto ao desempenho final através da métrica AUCarea, uma vez que a mesma foi utilizada durante o algoritmo. Os testes foram repetidos de forma igual e tiveram o mesmo resultado, i.e., estatisticamente DCIAPSO e DCIAPSO.M1 possuem desempenhos equivalentes em todas as bases, exceto na base *Yeast*.

Quanto à quantidade de protótipos gerados, os testes estatísticos comprovam que o DCIAPSO.M1 gerou mais protótipos que o DCIAPSO em todas as bases, exceto na base *Zoo*. Desta forma, há mais uma evidência da falha da suposição, uma vez que a mudança de métrica só permitiu desempenhos similares com uma quantidade maior de protótipos.

B.6 DCIAPSO.M2

Os experimentos conduzidos com o DCIASGSA.M4 demonstraram que é possível haver melhora no desempenho do classificador, no contexto de múltiplas classes desbalancea-

das, com a seleção de atributos. Entretanto os m melhores atributos não necessariamente formam o melhor subconjunto de atributos (LIN, 2012). Uma maneira de se formar um subconjunto de atributos que melhore o desempenho de classificação é utilizando a abordagem *wrapper* (MOAYEDIKIA et al., 2017). Nesta abordagem o melhor subconjunto é selecionado através de um processo de busca, e validado através da classificação dos dados com o subconjunto em questão (MAFARJA; MIRJALILI, 2018).

Nesta modificação, foi implementado o algoritmo de busca CSO para o processo de seleção de atributos, como sugerido em (GU; CHENG; JIN, 2018). O CSO foi originalmente proposto em (CHENG; JIN, 2015) como um algoritmo de otimização eficiente em larga escala.

Em vez de se atualizarem a partir de suas melhores posições e da melhor posição global, cada partícula é atualizada a partir de um competidor selecionado aleatoriamente. Em cada iteração o conjunto de partículas é dividido em dois grupos, os quais são comparados, de duas em duas partículas. O vencedor de cada comparação é passado para a próxima iteração, enquanto que a partícula perdedora passa por um processo de atualização de sua posição e velocidade, levando em conta a partícula vencedora. A velocidade e a posição das partículas são atualizadas com as seguintes equações:

$$v_l^{t+1} = R_1^t v_l^t + R_2^t(x_w^t - x_l^t) + \phi R_3^t(\bar{x}^t - x_l^t) \tag{B.4}$$

$$x_l^{t+1} = x_l^t + v_l^{t+1} \tag{B.5}$$

na qual R_1^t, R_2^t e R_3^t são três valores aleatórios entre 0 e 1, x_w^t e x_l^t são as partículas vencedora e perdedora, respectivamente, $\bar{x}^t$ é a posição média de todas as partículas na iteração t, e ϕ controla a influência de $\bar{x}^t$.

As partículas no CSO se *movimentam* em um espaço de busca contínuo. Entretanto, o espaço de busca da seleção de atributos é discreta. A solução proposta em (GU; CHENG; JIN, 2018) acrescenta um parâmetro de limiar λ que determina se um atributo é ou não selecionado.

Além do parâmetro λ, outra contribuição dos autores foi propor a utilização de um arquivo $\mathcal{H}$ para armazenar os valores de *fitness* das soluções. Esta proposição foi motivada pelo alto custo computacional das avaliações das soluções em bases com muitas dimensões. Desta forma, o *fitness* de novas soluções só são calculados após consulta no arquivo $\mathcal{H}$, evitando-se cálculos desnecessários de soluções já *visitadas*.

Os valores utilizados nos parâmetros são os seguintes: $\phi = 0,1$ e $\lambda = 0,5$. Cada partícula é um vetor no qual cada posição corresponde a um atributo. Uma vez que o CSO atribui valores contínuos para cada atributo, tais valores, um a um, são comparados ao parâmetro λ. Caso o valor seja maior, o atributo recebe o valor 1, significando que está selecionado. Do contrário, o atributo recebe o valor 0, significando que não faz parte daquele subconjunto.

Tabela 42 – Desempenho do DCIAPSO.M2 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8366±0,0131	245,6±62,9825	0,9264
Arrhythmia	0,7824±0,0487	67,4±14,3108	0,8136
Balance Scale	0,7702±0,0338	9,8±3,8987	0,9804
Car Evaluation	0,9352±0,0237	27±9,3005	0,9805
Contraceptive	0,6434±0,0112	13±5,8737	0,9890
Dermatology	0,9686±0,0408	15,4±5,4129	0,9474
E.coli	0,8943±0,0212	24,6±4,8270	0,9085
Gene	0,8595±0,0507	34,8±5,4955	0,9864
Glass	0,8797±0,0193	24±3,6742	0,8598
Hayes-Roth	0,8564±0,0667	13,6±5,9833	0,8938
Horse	0,7592±0,0310	16,2±3,5637	0,9447
Nursery	0,8320±0,0402	26,8±14,1669	0,9974
Page Blocks	0,8577±0,0280	28,6±5,5946	0,9935
Post Operative	0,6700±0,1026	16,8±6,9785	0,7667
Satimage	0,9228±0,0104	65,8±11,7771	0,9872
Shuttle	0,9699±0,0349	41,8±15,7385	0,9991
Soybean	0,9770±0,0072	48±28,8791	0,9048
Thyroid	0,9346±0,0618	9,6±3,0496	0,9983
Wine	0,9683±0,0161	9,4±4,0373	0,9340
Yeast	0,8276±0,0461	54,6±26,0634	0,9540
Zoo	0,9621±0,0210	13,8±5,9330	0,8292

Com esta modificação o DCIAPSO passa a ter os seguintes passos:

1. Divida os dados com *5-fold* DOB-SCV;
2. Normalize os dados do conjunto de treinamento e teste;
3. Selecione os atributos com CSO;
4. Execute DCIAPSO.

Na Tabela 42 são apresentados os resultados obtidos com esta nova modificação. Mais uma vez são apresentados as médias e desvios-padrões para as métricas MAUC e a quantidade de protótipos. É apresentada também a taxa de redução. As comparações estatísticas foram feitas entre esta modificação e o DCIAPSO.

A seleção de atributos com CSO teve por consequência a melhoria de desempenho em duas bases, *Gene* e *Horse*. Na base *Gene* houve um aumento de cerca de 10 pontos percentuais no desempenho com a métrica MAUC, ao mesmo tempo em que a quantidade

de protótipos gerados diminuiu, de cerca de 50 para 35 protótipos. Na base *Horse* a melhoria de desempenho foi menor, entretanto a quantidade de protótipos gerados continuou estatisticamente equivalente, logo, pode-se inferir que o algoritmo se tornou mais eficiente nesta base. Na base *Thyroid* também pode ser inferido que o algoritmo se tornou mais eficiente, pois ainda que não tenha havido melhora de desempenho, houve diminuição na quantidade de protótipos gerados.

B.7 DCIAPSO.M3

A modificação DCIAPSO.M3 foi implementada para que fosse verificada a consequência sobre o desempenho do algoritmo ao se modificar a forma de calcular o *fitness* durante a seleção de atributos.

Até então, cada subgrupo de atributos selecionados tinha seu *fitness* calculado com a métrica MAUC, após a classificação das instâncias do conjunto de treinamento. No DCIAPSO.M3 o *fitness* passou a ser calculado como a soma das distâncias dos centroides das classes. A partir disso, espera-se verificar se selecionar os atributos de modo a deixar as classes mais distantes umas das outras possa ter algum efeito positivo ou negativo sobre a classificação.

Na Tabela 43 são apresentados os resultados obtidos pela modificação DCIAPSO.M3. As comparações com o DCIAPSO comprovam o que pode ser percebido pela tabela. A modificação da função *fitness* não teve efeito sobre o desempenho do algoritmo. Entretanto, em duas bases, *Hayes-Roth* e *Thyroid* houve redução da quantidade de protótipos gerados.

Tabela 43 – Desempenho do DCIAPSO.M3 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8334±0,0087	285,6±34,3991	0,9144
Arrhythmia	0,7857±0,0396	58,2±11,7771	0,8390
Balance Scale	0,7755±0,0417	11,4±3,0496	0,9772
Car Evaluation	0,9261±0,0296	37±4,3589	0,9732
Contraceptive	0,6447±0,0429	11,8±2,3875	0,9900
Dermatology	0,9607±0,0171	20,6±9,8387	0,9296
E.coli	0,8960±0,0310	28,8±8,0436	0,8929
Gene	0,7697±0,0215	46,8±5,1672	0,9817
Glass	0,8437±0,0880	24±6,0415	0,8598
Hayes-Roth	0,7728±0,0745	10,4±2,5100	0,9187
Horse	0,6989±0,0254	21±9,7724	0,9283
Nursery	0,8268±0,0235	17,8±8,9554	0,9983
Page Blocks	0,8651±0,0213	31,8±11,5195	0,9927
Post Operative	0,5821±0,0797	17,8±2,5884	0,7528
Satimage	0,9250±0,0068	70,4±4,0373	0,9863
Shuttle	0,9539±0,0248	48,8±15,4984	0,9989
Soybean	0,9697±0,0193	43,8±16,3463	0,9131
Thyroid	0,8735±0,0242	13,2±2,9496	0,9977
Wine	0,9835±0,0137	5,8±1,9235	0,9593
Yeast	0,8282±0,0357	68,2±8,9833	0,9426
Zoo	0,9814±0,0129	10,2±3,4928	0,8738

B.8 DCIAPSO.M4

A utilização do CSO na seleção de atributos teve por consequência uma melhora de desempenho e eficiência do algoritmo. Ao mesmo tempo, existem outros algoritmos de seleção de atributos do tipo *wrapper* que podem ser testados. Desta forma será possível verificar se há alguma estratégia que possa melhorar ainda mais o desempenho do DCIAPSO. Nesta modificação, a seleção de atributos ocorre com o algoritmo WOA-CM (MAFARJA; MIRJALILI, 2018).

O WOA é um algoritmo evolucionário baseado no comportamento de alimentação de baleias jubarte (MIRJALILI; LEWIS, 2016; MAFARJA; MIRJALILI, 2018). Ele pertence aos algoritmos estocásticos baseados em população propostos em (MIRJALILI; LEWIS, 2016). Mais precisamente, o WOA simula a criação de uma rede de bolhas quando as baleias jubarte nadam próximo à superfície para caçar. Enquanto caçam, as baleias seguem um caminho no formato do número 6 (MIRJALILI; LEWIS, 2016).

Há duas fases no WOA. A primeira fase, de explotação, simula o rodeamento das

baleias sobre uma presa e o ataque com redes de bolhas. A segunda fase, de exploração, é uma busca aleatória por uma presa.

Na primeira fase, a movimentação das baleias ao redor da presa, com atualização de suas posições, é calculada da seguinte forma:

$$\vec{D} = |\vec{C} \cdot \vec{X}_t^* - \vec{X}_t| \tag{B.6}$$

$$\vec{X}_{t+1} = \vec{X}_t^* - \vec{A} \cdot \vec{D} \tag{B.7}$$

no qual t é a iteração corrente, X^* é a melhor solução até então, X é a solução corrente, $|\bullet|$ se refere ao valor absoluto, e o símbolo $\cdot$ se refere a uma multiplicação elemento por elemento. A e C são vetores de coeficiente calculados da seguinte forma:

$$\vec{A} = 2\vec{a}.\vec{r} - \vec{a} \tag{B.8}$$

$$\vec{C} = 2\vec{r} \tag{B.9}$$

no qual a decresce linearmente de 2 a 0 (Equação B.10) e r é um vetor aleatório nos valores [0, 1].

$$a = 2 - t\frac{2}{MaxIter} \tag{B.10}$$

no qual t é o número da iteração e $MaxIter$ é a quantidade máxima de iterações.

As posições das soluções são atualizadas de acordo com a posição da melhor solução (Equação B.7). O ajuste dos valores nos vetores A e C controla as áreas nas quais a solução pode ser localizada na vizinhança da melhor solução.

O movimento das baleias jubarte ao redor de uma presa é similar a uma espiral, i.e., um movimento circular, cujo raio vai diminuindo com o tempo. No WOA o movimento de aproximação é simulado na Equação B.8 ao se diminuir o valor de a, de acordo com a Equação B.10. A espiral a ser percorrida é calculada a partir da distância entre a solução corrente e a melhor solução, da seguinte forma:

$$\vec{X}_{t+1} = D' \cdot e^{bl} \cdot cos(2\pi l) + \vec{X}^*(t) \tag{B.11}$$

no qual $\vec{D}$ é a distância entre uma *baleia* X e uma presa ($\vec{D} = |\vec{X}_t^* - \vec{X}_t|$), b define a forma da espiral, e l é um número aleatório na faixa [-1, 1]. Os dois mecanismos, de aproximação e o caminho da espiral são escolhidos a partir de uma probabilidade p, um número aleatório na faixa [0, 1]. Caso $p < 0,5$, a Equação B.7 é utilizada, do contrário a Equação B.11.

Algoritmo 6: Pseudo-código do WOA

Gere População Inicial X_i $(i = 1, 2, ..., n)$
Calcule o valor objetivo de cada solução
enquanto $t < MaxIter$ **faça**
  para cada solução **faça**
    Atualize a, A, C, l e p
    se $p < 0,5$ **então**
      se $|A| < 1$ **então**
        Atualize a posição da solução corrente com a Equação B.7
      senão se $|A| > 1$ **então**
        Selecione uma solução aleatória X_{rand}
        Utilize a Equação B.13
      fim do se
    senão
      Atualize a posição da solução corrente com a Equação B.11
    fim do se
  fim do para
  Verifique se qualquer solução está além do espaço de busca e corrija
  Calcule o fitness *de cada solução*
  Se há alguma melhor solução global *atualize* X^*
  $t = t + 1$
fim do enquanto
retorne X^*

Na segunda fase uma solução aleatória é escolhida e utilizada para *forçar* a solução corrente a se mover para longe da melhor solução encontrada até então. Para isto é utilizado um vetor com valores aleatórios maiores e menores que 1:

$$\vec{D} = |\vec{C} \cdot \vec{X}_{rand} - \vec{X}_t| \tag{B.12}$$

$$\vec{X}_{t+1} = \vec{X}_{rand} - \vec{A} \cdot \vec{D} \tag{B.13}$$

no qual $\vec{X}_{rand}$ é uma solução aleatória, dentre as existentes. O pseudo-algoritmo do WOA, como apresentado em (MAFARJA; MIRJALILI, 2018), é adaptado no Algoritmo 6.

A adaptação do WOA para a seleção de atributos, de início, é similar ao CSO. As soluções são um vetor unidimensional cuja quantidade de elementos equivale à quantidade total de atributos de uma base de dados. Para cada elemento do vetor, o número 1 é utilizado para definir que o atributo corrente é selecionado, enquanto o número 0 define que o atributo selecionado será descartado.

A função de aptidão utilizada no WOA-CM foi desenvolvida com o intuito de balancear a quantidade de atributos selecionados e o desempenho de classificação obtido com o uso

de tais atributos. Esta função é calculada da seguinte maneira:

$$fitness = \alpha\gamma_R(D) + \beta\frac{|R|}{|C|} \tag{B.14}$$

no qual $\gamma_R(D)$ é o erro de classificação, $|R|$ é a cardinalidade da solução corrente, i.e., a quantidade de atributos selecionados, $|C|$ é a quantidade total de atributos da base de dados, α e β são dois parâmetros para o ajuste da importância dada ao desempenho de classificação e à quantidade de atributos selecionados, respectivamente. O valor atribuído a α é 0,99 enquanto β recebe o valor de 0,01, ou seja, $1 - \alpha$.

A atualização das soluções, antes calculadas através das Equações B.7 e B.13, passa a acontecer com os operadores de mutação e *crossover* (MAFARJA; MIRJALILI, 2018). O operador de mutação depende de uma taxa de mutação r e funciona da seguinte forma. Para cada elemento da solução corrente um número aleatório na faixa [0, 1] é selecionado. Caso o número seja maior que a taxa r, o valor do elemento é trocado. Ou seja, se o atributo em questão estiver selecionado, passa a ser descartado, e vice-versa. O valor da taxa de mutação é decrementado linearmente de 0,9 para 0, de acordo com a iteração t:

$$r = 0,9 + \frac{-0,9 * (t-1)}{MaxIter - 1} \tag{B.15}$$

O operador de *crossover* também é simples. Após selecionadas duas soluções, a nova solução é construída como segue. Para cada elemento do vetor um número aleatório p na faixa [0, 1] é selecionado. Caso $p \geq 0,5$ o valor da primeira solução é copiado. Caso contrário, o valor da segunda solução é copiado. A aplicação dos dois operadores acontece como mostrado no Algoritmo 7.

A implementação do WOA-CM foi adaptada do original, encontrado eletronicamente no endereço <https://www.mathworks.com/matlabcentral/fileexchange/55667-the-whale-optimization-algorithm>. A adaptação foi realizada para ajustar alguns nomes de variáveis e outros detalhes necessários para a execução do algoritmo junto ao DCIAPSO. É importante frisar também que a *memória* para guardar o *fitness* de soluções já visitadas, como apresentado no DCIAPSO.M2, foi utilizado junto ao WOA-CM. Por fim, o cálculo do *fitness* levou em conta a métrica MAUC, em vez da Acurácia, como no algoritmo original.

Na Tabela 44 são apresentados os resultados obtidos com o DCIAPSO.M4. A base *Shuttle* é a única ausente devido ao tempo de execução do algoritmo na mesma ser bastante elevado. A seleção de atributos, por si só, já é bastante custosa computacionalmente (GU; CHENG; JIN, 2018). Entretanto, mesmo com a utilização de memória para que se diminua o custo, o WOA-CM ainda se mostrou custoso o suficiente para que não fosse possível obter os resultados da maior das bases.

A comparação estatística do DCIAPSO.M4 ao DCIAPSO evidencia mais uma vez que a seleção de atributos melhora o desempenho e eficiência de classificação do algoritmo.

Algoritmo 7: Pseudo-código do WOA-CM

Gere População Inicial X_i $(i = 1, 2, ..., n)$
Calcule o valor objetivo de cada solução
Melhor solução: X^*
enquanto $t < MaxIter$ **faça**
  para cada solução **faça**
    Calcule a taxa de mutação como na Equação B.15
    Atualize a, A, C, l e p
    se $p < 0,5$ **então**
      se $|A| < 1$ **então**
        Aplique mutação em X^* *obtendo* X^{Mut}
        Atualize a solução corrente através do crossover *com* X^{Mut}
      senão se $|A| > 1$ **então**
        Selecione uma solução aleatória X_{rand}
        Aplique mutação em X_{rand} *obtendo* X^{Mut}
        Atualize a solução corrente através do crossover *com* X^{Mut}
      fim do se
    senão
      Atualize a posição da solução corrente com a Equação B.11
    fim do se
  fim do para
  Verifique se qualquer solução está além do espaço de busca e corrija
  Calcule o fitness *de cada solução*
  Se há alguma melhor solução global *atualize* X^*
  $t = t + 1$
fim do enquanto
retorne X^*

Na base *Thyroid*, observando-se a métrica MAUC, houve um salto de desempenho médio de 0,8712 para 0,9510, enquanto a quantidade de protótipos gerados foi de 16,8 no DCIAPSO e 11,4 na modificação corrente. Na base *Abalone* também houve redução da quantidade média de protótipos gerados, de 273,4 para 236. Em todas as outras bases, tanto o desempenho quanto a geração média de protótipos permaneceu estatisticamente equivalente.

A base *Thyroid* possui uma característica de ter uma classe muito grande, enquanto as demais podem ser consideradas minoritárias. A classe majoritária nesta base é responsável por cerca de 93% de todas as instâncias. Neste quesito, somente a base *Page-Blocks* pode ser comparável dentre as bases de dados avaliadas neste trabalho, pois também possui uma classe majoritária *dominante*, e as demais podem ser consideradas classes minoritárias.

Entretanto, algumas diferenças entre essas bases podem explicar porque houve melhoria de desempenho e eficiência em somente uma dessas duas bases. A primeira diferença é a quantidade de classes. A base *Thyroid* possui três, enquanto a base *Page-Blocks* possui quatro. A concentração da classe majoritária na primeira é maior, com cerca de 93%, enquanto na segunda é de aproximadamente 90%. A quantidade de atributos também difere,

Tabela 44 – Desempenho do DCIAPSO.M4 utilizando das métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8338±0,0131	236±18,8944	0,9293
Arrhythmia	0,7946±0,0300	59,2±11,3666	0,8363
Balance Scale	0,7963±0,0483	9,8±4,4385	0,9804
Car Evaluation	0,9293±0,0375	35,4±5,1284	0,9744
Contraceptive	0,6558±0,0354	15±4,0620	0,9873
Dermatology	0,9634±0,0339	9,6±1,8166	0,9672
E.coli	0,9000±0,0433	26,2±4,6583	0,9025
Gene	0,7621±0,0147	46,2±5,0695	0,9819
Glass	0,8659±0,0702	25,2±10,1833	0,8528
Hayes-Roth	0,8692±0,0415	11,4±4,0988	0,9109
Horse	0,7049±0,0698	15±4	0,9488
Nursery	0,8028±0,0363	39±27,3222	0,9962
Page Blocks	0,8474±0,0282	34,8±8,4083	0,9921
Post Operative	0,6294±0,0876	17,8±3,3466	0,7528
Satimage	0,9248±0,0096	72,8±8,6718	0,9859
Soybean	0,9729±0,0164	39,8±12,4579	0,9210
Thyroid	0,9510±0,0405	11,4±3,9115	0,9980
Wine	0,9834±0,0126	6±3,4641	0,9579
Yeast	0,8182±0,0361	65,2±10,8950	0,9451
Zoo	0,9705±0,0088	13±8,2765	0,8391

de 21 para 10. Por fim, a separabilidade entre as classes é maior na base *Page-Blocks*, de acordo com os seus valores de F_{gen}.

A estratégia geral adotada para geração de protótipos já pode estar explorando o máximo possível dos benefícios que poderia extrair da base *Page-Blocks*. Em outras palavras, possivelmente não há melhor conjunto de atributos a ser selecionado que possa, de alguma forma, deixar as classes mais discriminadas entre si. É possível também que a forma e separação entre as classes não possam ser melhor abordadas sem haver grande modificação no algoritmo. Entretanto, a base *Thyroid*, com uma menor quantidade de classes, as quais estão bastante próximas umas das outras, ainda permite a obtenção de melhores desempenhos ao se modificar alguns detalhes do algoritmo.

Uma comparação empírica do DICAPSO.M4 com o DCIAPSO.M2 mostra que ambos conseguem prover alguma melhoria em bases diferentes. Em relação à base *Thyroid*, o CSO influenciou para que o desempenho de classificação fosse mantido, porém com uma quantidade menor de protótipos. O WOA-CM, por outro lado, foi mais bem sucedido, conseguindo influenciar também a melhoria de desempenho. Possivelmente, algum algo-

ritmo que selecione, para cada base, a melhor técnica de seleção de atributos possa obter melhores resultados de forma geral.

B.9 DCIAPSO.M5

Moayedikia *et al.* (MOAYEDIKIA et al., 2017) argumentam que a seleção de atributos baseada em função de perda, i.e., levando em consideração a taxa de erro ou o desempenho do algoritmo, nem sempre resultam em melhorias para o classificador. Diante disso, os autores sugerem ordenar os atributos de acordo com sua dependência em relação às classes. Esta informação deve então ser utilizada para selecionar o subconjunto de atributos. A ordenação dos atributos e a maneira como a informação é utilizada faz parte da estratégia do algoritmo SYMON.

O SYMON, assim como o WOA-CM, é um algoritmo de seleção de atributos do tipo *wrapper*. Segundo os autores, é um algoritmo que lida bem com bases de dados com alta dimensionalidade, conseguindo encontrar o melhor subconjunto mesmo quando vários atributos são igualmente bem ranqueados (MOAYEDIKIA et al., 2017). A ordenação dos atributos é feita com Incerteza Simétrica (*Symmetrical Uncertainty*), e o processo de busca dos subconjuntos é feita com Busca Harmônica (*Harmonic Search*).

O Pseudo-código do SYMON é apresentado, traduzido do original, no Algoritmo 8. O primeiro passo do algoritmo é ranquear todos os atributos usando Incerteza Simétrica (linha 3), antes da inicialização da busca harmônica. A Incerteza Simétrica já se mostrou efetiva na seleção de atributos em bases de dados de larga escala (KANNAN; RAMARAJ, 2010; RUIZ et al., 2012; MOAYEDIKIA et al., 2017). É baseada em entropia, e mede a incerteza de uma variável aleatória x em relação a outra variável y:

$$H(x) = -\sum P(x_i) log_2(P(x_i)) \tag{B.16}$$

$$H(x|y) = -\sum_i P(y_i) \sum_j P(x_i|y_i) log_2(P(x_i|y_j)) \tag{B.17}$$

no qual $P(x_i)$ é a probabilidade *a priori* de todos os valores de x e $P(x_i|y_i)$ é a probabilidade *a posteriori* de x dado y. A partir das Equações B.16 e B.17 é possível obter o ganho de informação:

$$\mathcal{G}(x|y) = H(x) - H(x|y) \tag{B.18}$$

O ganho de informação $\mathcal{G}$ é uma medida simétrica para cada par x, y, logo pode ser utilizado para determinar a correlação entre seus elementos. Entretanto, é necessário que

Algoritmo 8: Pseudo-código do SYMON

Input: F, conjunto de atributos
C, conjunto de rótulos de classes
NI, quantidade de iterações
HMS, tamanho da memória harmônica
HMCR, taxa de consideração da memória harmônica
PAR_{max}, taxa máxima de ajuste de tom
PAR_{min}, taxa mínima de ajuste de tom
Output: HM, vetores de soluções otimizadas na memória harmônica
$w = CalculateSU(F, C)$;
Inicializar();
para $t = 1$ até NI **faça**
  para $f \in$ F **faça**
    $R_f \leftarrow$ número aleatório;
    se $R_f >$ HMCR **então**
      Selecione aleatoriamente um vetor v_r de HM;
      NHV[f] $\leftarrow v_r[f]$;
      $R_p \leftarrow$ número aleatório;
      $P_f \leftarrow$ PAR(t);
      se $P_f < R_p$ **então**
        NHV[f] $\leftarrow \overline{NHV[f]}$;
      fim do se
    senão
      $\phi \leftarrow$ número aleatório;
      se $\phi >$ 0,5 **então**
        NHV[f] $\leftarrow$ 1;
      senão
        NHV[f] $\leftarrow$ 0;
      fim do se
    fim do se
  fim do para
  NHV = VectorTune(NHV, w, r, d);
  se f(NHV) $> f(v)$ **então**
    HM = HM - $\{v\} \cup \{$NHV$\}$;
  fim do se
fim do para

os valores obtidos através da Equação B.18 estejam normalizados para que haja uma comparação significativa (MOAYEDIKIA et al., 2017). Formalmente:

$$\mathcal{S}(x, y) = \frac{2\mathcal{G}(x|y)}{H(x) - H(y)} \tag{B.19}$$

no qual $\mathcal{S} = 1$ implica em x e y serem completamente correlacionados, enquanto $\mathcal{S} = 0$ implica que x e y são independentes.

No SYMON a variável x corresponde ao atributo em consideração, enquanto y é o rótulo da classe. Desta forma, a Equação B.19 calcula a incerteza simétrica de um atributo

f_i em relação a uma única classe c. Através da Equação B.20 é possível calcular o peso de um atributo f_i sobre todas as classes:

$$\mathcal{M}(f_i, c) = \frac{\mathcal{S}(f_i|c)}{\sum_j \mathcal{S}(f_j|c)} \tag{B.20}$$

Na linha 3 do Algoritmo 8, F é o conjunto de todos os atributos e C é o conjunto de todas as classes. O *output* do algoritmo de incerteza simétrica é um conjunto de pesos de cada atributo, calculado de acordo com a Equação B.20.

Após o primeiro passo, a inicialização do SYMON (linha 4) consiste na criação de vetores binários e aleatórios, além do cálculo do *fitness* de cada vetor. O *fitness* é calculado a partir do desempenho de um classificador com a métrica *G-Mean* ou AUC.

Nas linhas 5 a 29 acontece a busca harmônica. Este algoritmo de busca foi originalmente proposto por Geem (GEEM, 2008). Ele imita o processo de improvisação musical, e pode ser dividido em cinco passos principais (GEEM, 2008; MOAYEDIKIA et al., 2017):

- **Inicialização:** neste primeiro passo são definidos os parâmetros do algoritmo. Os parâmetros são: tamanho da memória harmônica (*Harmony Memory Size* — HMS), taxa de consideração da memória harmônica (*Harmony Memory Consideration Rate* — HMCR), quantidade de iterações (*Number of Iterations* — NI) e a taxa de ajuste do tom (*Pitch Adjustment Rate* – PAR);
- **Improvisação:** neste passo é criado um novo vetor harmônico (*New Harmony Vector* — NHV). O novo vetor possui características próprias e também características herdadas de outros vetores já criados;
- **Avaliação:** o NHV é avaliado;
- **Substituição:** Após a avaliação do NHV, há uma comparação com os vetores existentes na memória harmônica (*Harmony Memory* — HM). O pior vetor encontrado na HM é substituído pelo NHV, caso seu desempenho seja melhor;
- **Checagem de Critério de Parada:** Ao fim de cada iteração é verificado se o critério de parada foi atingido.

Os valores dos parâmetros, de acordo com o código fonte[1], são 0,65 para HMCR, 200 para HMS, 0,9 para PAR_{max}, 0,5 para PAR_{min} e 200 para NI. Os vetores da memória harmônica são criados na inicialização. O PAR é calculado da seguinte forma:

$$PAR(t) = \frac{t}{NI} \times (PAR_{max} - PAR_{min}) \tag{B.21}$$

no qual t é a iteração corrente.

[1] Disponível em <http://moayedikiaalireza.wixsite.com/home/code-of-papers>.

No SYMON, o NHV é construído da seguinte forma. Para cada atributo (linha 6) um número aleatório é selecionado (linha 7). Caso esse número seja maior que HMCR, um vetor aleatório da memória é escolhido, e o valor assinalado para o atributo corrente é copiado para o atributo do NHV. Caso contrário, o atributo corrente é assinalado 0 ou 1 aleatoriamente (linhas 16 a 23). Entretanto, os valores copiados podem ser invertidos, caso o valor de PAR seja maior que um número aleatório (linhas 11 a 15).

Após a criação do NHV, o mesmo passa por um processo de ajuste (linha 25). Os parâmetros do *VectorTune* são o NHV, o conjunto de pesos dos atributos, fator de ondulação, e tamanho desejado do subconjunto de atributos, respectivamente. O fator de ondulação (*ripple factor*) determina a combinação de atributos a ser considerado, e no código fonte é configurado com o valor 1. O tamanho desejado do subconjunto é calculado da seguinte forma:

$$d = floor(\frac{c \times F}{5}) \tag{B.22}$$

no qual d é o tamanho desejado, c é um coeficiente configurado com o valor 1, e F é a quantidade total de atributos. A função $floor$ retorna o menor número inteiro a partir do valor obtido. Por exemplo, para o número 4,2 a função retorna o número 4. Ainda, caso aconteça de o valor de d resultar em 0, o mesmo é alterado para 1, ou seja, o tamanho desejado do subconjunto será de 1 atributo.

O ajuste do NHC segue dois princípios, ou definições (MOAYEDIKIA et al., 2017): (1) um atributo é mais significativo se (i) sua inclusão no subconjunto resulta em melhoria de desempenho quando comparado com outros atributos e (ii) sua exclusão do subconjunto reduz significativamente o desempenho quando comparado à exclusão de outros atributos no mesmo subconjunto; (2) um atributo é menos significante quando sua exclusão do subconjunto não causa redução significativa do desempenho quando comparado com a exclusão de outros atributos no mesmo subconjunto.

Os dois princípios são implementados na forma de duas operações: *Ripple_Add()* e *Ripple_Rem()*. Essas duas operações utilizam os parâmetros fator de ondulação r, e tamanho desejado do subconjunto d.

Todos os atributos são divididos em dois subconjuntos: F_s que reúne os atributos selecionados e F_u, o qual reúne os atributos não selecionados. A operação *Ripple_Add(r)* retira os r atributos mais significativos de F_u e os adiciona em F_s, ao mesmo tempo em que retira os $r - 1$ atributos menos significativos de F_s. A operação *Ripple_Rem(r)* remove os r atributos menos significativos de F_s enquanto adiciona os $r-1$ atributos mais significativos de F_u em F_s.

O processo de adição e remoção de atributos é iniciado a partir dos atributos selecionados e não selecionados com o mesmo peso. Se adicionar e remover os atributos com o mesmo peso não satisfizer o critério de tamanho do subconjunto, o restante dos atributos passam a ser vistos como numa fila para serem incluídos ou excluídos. Dependendo do

tamanho do subconjunto F_s e do parâmetro d, há três possíveis cenários (MOAYEDIKIA et al., 2017):

- O tamanho desejado do subconjunto é igual à quantidade de atributos selecionados. O subcojunto F_s é modificado com *Ripple_Add(r)* e *Ripple_Rem(r)*;
- O tamanho desejado do subconjunto é maior que a quantidade de atributos selecionados. Neste caso, aplica-se *Ripple_Add(r)* sobre o subconjunto F_s;
- O tamanho desejado do subconjunto é menor que a quantidade de atributos selecionados. Neste caso, aplica-se *Ripple_Rem(r)* sobre o subconjunto F_s.

Terminado o ajuste em NHV, seu *fitness* é comparado ao da pior solução presente em HM. Caso seu desempenho seja melhor, NHV substitui a pior solução (linhas 26 a 28).

O SYMON foi adicionado ao DCIAPSO para ser executado após a normalização dos dados. O código utilizado corresponde a uma adaptação do código fonte original. De forma similar ao WOA-CM, a adaptação consistiu apenas em algumas pequenas mudanças, como nomes de variáveis, por exemplo. Contudo, é importante frisar que o cálculo do *fitness* foi modificado. No algoritmo original o cálculo era feito em uma perspectiva de base binária com a métrica *G-Mean*. Com o DCIAPSO passou a ser utilizada a métrica MAUC, apropriada para múltiplas classes desbalanceadas.

Na Tabela 45 são apresentados os resultados obtidos com o DCIAPSO.M5. Outra vez a base *Shuttle* não foi incluída, devido ao alto custo computacional do SYMON. A comparação realizada com o DCIAPSO mostra que a utilização do SYMON para redução de atributos teve como consequência uma piora de desempenho em quase todas as bases de dados utilizadas.

Alguns fatores podem ter contribuído para tal resultado. O primeiro é que o SYMON, apesar de poder ser usado em uma base com múltiplas classes, foi desenvolvido e testado em bases binárias. Outro fator é que, das bases utilizadas neste estudo, apenas uma possui uma grande quantidade de atributos em relação as outras. Esta base é a *Arrhythmia*, que possui 260 atributos. Entretanto, o trabalho onde o SYMON foi proposto (MOAYEDIKIA et al., 2017) utiliza bases cuja quantidade de atributos varia de 2000 a mais de 24 mil. É possível que este algoritmo obtenha bons resultados somente em bases com altíssimas quantidades de atributos. Por fim, o algoritmo é bastante dependente dos parâmetros d e r, de modo que é necessário haver um processo de ajuste dos mesmos para a obtenção de melhores resultados.

Tabela 45 – Desempenho do DCIAPSO.M5 utilizando as métricas MAUC, Geração e Redução de Instâncias

Nome	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,6883±0,0255	53,8±6,0166	0,9839
Arrhythmia	0,7384±0,0268	97,4±9,3167	0,7306
Balance Scale	0,6410±0,0086	4,4±2,0736	0,9912
Car Evaluation	0,7123±0,0184	16,4±7,4699	0,9881
Contraceptive	0,5804±0,0119	9,4±3,3615	0,9920
Dermatology	0,6050±0,0295	18,6±12,2597	0,9365
E.coli	0,7663±0,0401	72,4±31,9500	0,7307
Gene	0,5314±0,0196	8,8±3,9623	0,9966
Glass	0,6577±0,0251	18±17,9444	0,8949
Hayes-Roth	0,5946±0,0354	5±1,0000	0,9609
Horse	0,5801±0,0323	4,6±1,1402	0,9843
Nursery	0,6698±0,0350	25±7,8740	0,9976
Page Blocks	0,5672±0,0342	9±1,2247	0,9979
Post Operative	0,6610±0,0646	10±8,5732	0,8611
Satimage	0,7829±0,1443	15,8±4,8683	0,9969
Soybean	0,7599±0,0159	64,6±31,0210	0,8718
Thyroid	0,5394±0,0400	6,8±1,7889	0,9988
Wine	0,8495±0,0095	10,2±3,4205	0,9284
Yeast	0,6939±0,0768	46,8±36,7110	0,9606
Zoo	0,7701±0,0546	14±2,7386	0,8267

B.10 DCIAPSO.M6

Apesar dos resultados ruins, antes de o SYMON ser descartado, uma modificação foi testada sobre o DCIAPSO.M5, mais especificamente em como calcular o *fitness*. A motivação foi verificar se outra maneira de calcular o *fitness* durante a seleção de atributos poderia ter um impacto positivo sobre o desempenho do algoritmo.

O DCIAPSO.M6 tem inspiração no algoritmo DCIAPSO.M3. Naquela modificação foi verificada a influência sobre o desempenho do algoritmo ao se selecionar os atributos de modo a deixar as classes mais distantes umas das outras levando-se em conta os centroides. A ideia da sexta modificação é similar, e o *fitness* passou a ser calculado da seguinte forma:

$$fitness = \sum((mean\_outer - std\_outer) - (mean\_inner + std\_inner)) \qquad \text{(B.23)}$$

O $mean\_outer$ e std_outer são calculados como segue. Para cada instância de cada classe, é calculada sua distância euclidiana para todas as instâncias de outras classes (distância externa). A partir de então é possível calcular uma média aritmética e desvio-padrão

dessas distâncias para cada classe. Os valores são guardados em dois vetores chamados de *mean_outer* e *std_outer*. Os vetores *mean_inner* e o *std_inner* são preenchidos de forma similar, com a diferença de que as distâncias são aquelas entre cada instância e todas as outras da mesma classe (distância interna).

As equações B.24 e B.25 apresentam formalmente o cálculo das distâncias externas e internas, respectivamente. Nestas equações, C é quantidade total de classes, $d(x_i, x_j)$ é a distância euclidiana entre duas instâncias, x_{ci} é a instância i da classe c, e n_c é a quantidade total de instâncias da classe c.

$$\sum_{k,l=1}^{C} d(x_{ki}, x_{lj}), \quad k \neq l, \quad i = 1...n_k, \quad j = 1...n_l \tag{B.24}$$

$$\sum_{c=1}^{C} d(x_{ci}, x_{cj}), \quad i, j = 1...n_c, \quad i \neq j \tag{B.25}$$

Normalmente as distâncias externas são maiores que as distâncias internas. Quanto mais densa é uma classe, ou seja, quanto menor a variação dos valores dos atributos das instâncias de uma mesma classe, maior a probabilidade de todas as distâncias internas serem menores que as distâncias externas.

As distâncias externas tendem a ter um alto valor de desvio-padrão, principalmente em bases com múltiplas classes, e também quando as classes são pouco densas. Subtrair os desvios-padrões das distâncias externas de suas médias possivelmente produz um valor baixo, podendo até ser negativo. Por outro lado, somar o desvio-padrão pode produzir um valor bem alto, principalmente quando há classes pouco densas.

Para um melhor entendimento a Figura 49 apresenta um exemplo de base com dois atributos e três classes, as quais são representadas através de suas instâncias como círculo, triângulo e losango. Observa-se que a classe círculo é a mais densa, seguida pela classe losango, enquanto a classe triângulo é pouco densa, ou esparsa. Levando-se em consideração o atributo da abscissa, representado como x, as classes círculo e losango contribuem bastante para médias aritméticas e desvios-padrões com valores baixos, enquanto a classe triângulo possui valores maiores. Entretanto, ao se levar em consideração somente o outro atributo, representado pela ordenada y, tanto a classe losango quanto a triângulo terão maiores valores de média aritmética e desvio-padrão.

Portanto, o cálculo do *fitness* como apresentado na Equação B.23, pressiona a busca para encontrar um conjunto de atributos que mapeie as classes de forma densa. A subtração dos desvios-padrões das distâncias externas porém, sacrifica a separabilidade entre as mesmas.

Os resultados obtidos com o DCIAPSO.M6 são apresentados na Tabela 46. A modificação do cálculo de *fitness* durante a seleção de atributos resultou em piora de desempenho. As comparações feitas entre a modificação corrente e os algoritmos DCIAPSO e DCIAPSO.M5 mostram que houve piora até mesmo em relação à modificação anterior.

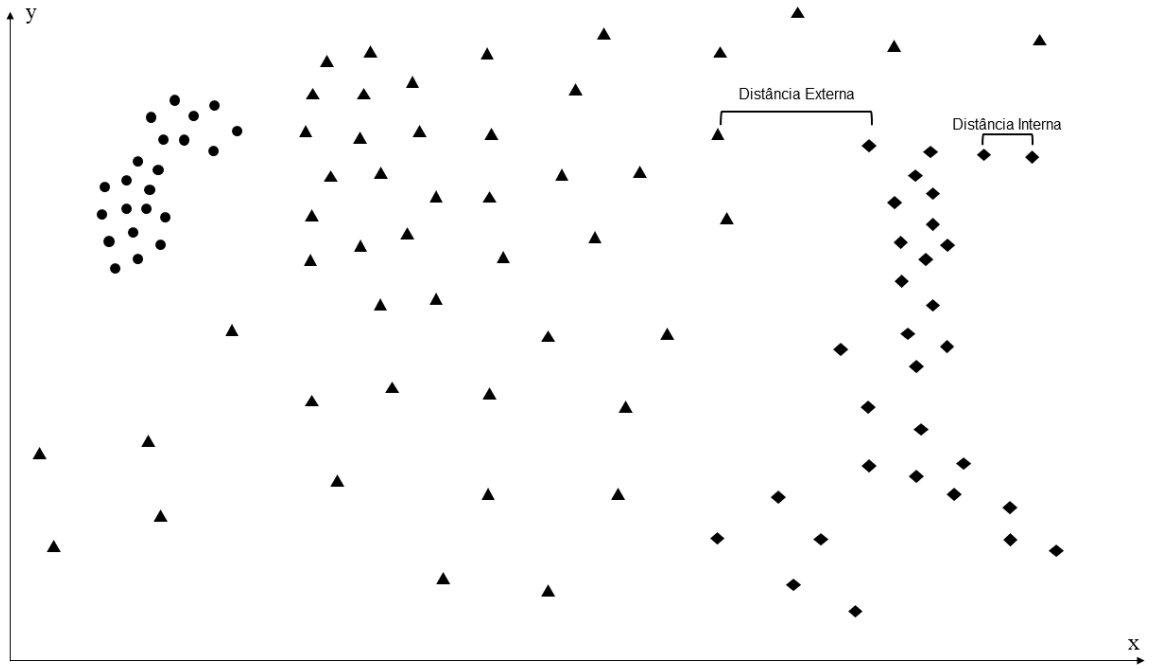

Figura 49 – Exemplo de base de dados com classes densa e pouco densa, além da distância interna e externa entre elas.

Existem duas possíveis explicações para a piora de desempenho. A primeira é sobre o uso do SYMON para seleção de atributos, de modo que, independentemente das modificações, é possível que só haja melhoria com ajustes nos parâmetros deste algoritmo. A segunda explicação é a má concepção do cálculo de *fitness*, ou seja, a fórmula apresentada na Equação B.23 deve ser refeita, ou mesmo a ideia ser descartada. As próximas três modificações verificam estas suposições.

Tabela 46 – Desempenho do DCIAPSO.M6 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,6294±0,0213	44±2,9155	0,9868
Arrhythmia	0,7556±0,0385	99,8±11,7771	0,7240
Balance Scale	0,5667±0,0549	6,2±1,3038	0,9876
Car Evaluation	0,6134±0,0726	12,4±7,6026	0,9910
Contraceptive	0,5647±0,0360	8±3,5355	0,9932
Dermatology	0,6129±0,0544	14,6±7,6026	0,9501
E.coli	0,7896±0,0473	80,6±12,4016	0,7001
Gene	0,5148±0,0063	7±2,8284	0,9973
Glass	0,6692±0,0596	12,2±1,7889	0,9287
Hayes-Roth	0,6373±0,0541	5±1,5811	0,9609
Horse	0,5734±0,0540	5,6±2,6077	0,9809
Nursery	0,5348±0,0291	11,2±2,3875	0,9989
Page Blocks	0,5590±0,0342	9,6±0,5477	0,9978
Post Operative	0,6138±0,0531	9±3,8079	0,8750
Satimage	0,5276±0,0049	12,2±2,7749	0,9976
Soybean	0,6873±0,0503	37±19,6214	0,9266
Thyroid	0,5431±0,0230	9,8±6,9065	0,9983
Wine	0,8581±0,0214	10,4±4,0373	0,9270
Yeast	0,6452±0,0390	18,8±3,5637	0,9842
Zoo	0,7809±0,0880	14,6±2,8810	0,8193

B.11 DCIAPSO.M7

O DCIAPSO.M7 tem como motivação a verificação do cálculo do *fitness*, como na Equação B.23, em outro algoritmo de seleção de atributos, além do SYMON (MOAYEDIKIA et al., 2017). Dos três algoritmos de seleção de atributos testados, o CSO foi o que obteve os melhores resultados até então. Portanto, esta modificação é uma versão do DCIAPSO.M6, entretanto utilizando o CSO no lugar do SYMON.

Os resultados desta modificação podem ser consultados na Tabela 47. O DCIAPSO.M7 foi comparado com o DCIAPSO e DCIAPSO.M5. Em relação ao primeiro algoritmo, o DCIAPSO.M7 continuou com a tendência de ser um algoritmo com menor desempenho, com o agravante de ter aumentado a quantidade de protótipos gerados em quase metade das bases.

Em relação ao DCIAPSO.M5 houve piora de desempenho em cinco bases de dados, e melhora de desempenho em outras cinco bases. As bases em que houve piora de desempenho foram: *Balance Scale*, *Car Evaluation*, *Contraceptive*, *Glass* e *Satimage*. Por outro lado, as bases em que houve melhoria de desempenho foram: *Dermatology*, *E.coli*,

Tabela 47 – Desempenho do DCIAPSO.M7 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,6357±0,0362	41,4±5,1284	0,9876
Arrhythmia	0,7301±0,0484	25,8±3,5637	0,9287
Balance Scale	0,5317±0,0159	4±0	0,9920
Car Evaluation	0,5843±0,0183	8,4±2,0736	0,9939
Contraceptive	0,5209±0,0152	6,4±2,0736	0,9946
Dermatology	0,9563±0,0098	11±4,1231	0,9624
E.coli	0,8817±0,0360	24±4,7434	0,9107
Gene	0,5262±0,0084	5,6±1,8166	0,9978
Glass	0,6060±0,0219	13,4±3,2094	0,9217
Hayes-Roth	0,5855±0,0481	5,6±2,0736	0,9563
Horse	0,5832±0,0415	5,4±1,8166	0,9816
Nursery	0,7079±0,1184	14,6±3,5071	0,9986
Page Blocks	0,5806±0,0313	8±2,5495	0,9982
Post Operative	0,6894±0,1255	7,2±1,7889	0,9000
Satimage	0,5183±0,0064	10±3,0000	0,9981
Soybean	0,9814±0,0061	60,4±49,0337	0,8802
Thyroid	0,5345±0,0137	5,6±1,8166	0,9990
Wine	0,9860±0,0182	5,2±1,3038	0,9635
Yeast	0,6301±0,0412	19±2,3452	0,9840
Zoo	0,9729±0,0084	10±2,3452	0,8762

Soybean, *Wine* e *Zoo*.

Ao se observar as características das bases foi possível encontrar algumas correlações lineares que podem explicar tanto a queda quanto o ganho de desempenho. A quantidade de instâncias e atributos, além do quão próximas são as classes (F_{gen}), estão correlacionadas à piora de desempenho. Em outras palavras, as bases com mais instâncias e atributos, com classes mais próximas, tiveram maior queda de desempenho. Entretanto, quando excluída a base *Satimage* dos cálculos de correlação, é possível perceber que esta base se mostra como um *outlier* que enviesa os resultados.

Observando-se novamente as bases nas quais houve queda de desempenho, excluindo-se a base *Satimage*, é possível perceber que em relação à quantidade de atributos, a correlação é alta e inversa. Ou seja, nas bases com maior quantidade de atributos houve menor queda de desempenho. Em relação à quantidade de instâncias e F_{gen} as correlações podem ser vistas como fraca a moderada, positiva e negativa, respectivamente. Em outras palavras, uma maior quantidade de instâncias algumas vezes pode ser associado a uma maior queda de desempenho, e um F_{gen} menor (ou seja, classes mais próximas entre si)

também pode ser associado, em alguns casos, a uma maior queda de desempenho.

A explicação para a base *Satimage* ter sido uma exceção é que os valores originais de seus atributos são bastante diversos. Portanto, após seus dados serem normalizados, as distâncias internas e externas acabam por ter valores que prejudicam o processo de seleção de atributos.

As características das bases em que houve aumento de desempenho também possuem correlações lineares. Quanto ao F_{gen}, a tendência observada anteriormente se confirma neste caso com uma forte correlação negativa. Ou seja, quanto mais distantes são as classes, maior foram os aumentos de desempenho. O IR e o MIR também possuem correlação negativa com o aumento de desempenho, porém, moderado, indicando que em alguns casos, quanto menor seu valor (classes mais balanceadas), maior o aumento de desempenho. Por fim, a quantidade de atributos também possui correlação com o aumento de desempenho. A maior quantidade de atributos neste caso pode ser benéfico devido a uma maior variedade dos mesmos. Desta forma, é possível que haja maiores chances de se encontrar um subconjunto que discrimine melhor as classes.

Os resultados obtidos com o DCIAPSO.M7 indicam, portanto, que o SYMON tem grande influência na diminuição do desempenho. O cálculo de *fitness* como na Equação B.23 também influencia na diminuição do desempenho, porém com menor impacto, já que a queda de desempenho em várias bases com o DCIAPSO.M5 foi praticamente revertido. Ou seja, ficaram próximos ao desempenho obtido com o DCIAPSO.

B.12 DCIAPSO.M8 E DCIAPSO.M9

Os algoritmos DCIAPSO.M8 e DCIAPSO.M9 são modificações diretas no DCIAPSO.M7. Mais especificamente, são dois ajustes na Equação (B.23), durante a seleção de atributos. Os ajustes foram feitos da seguinte forma:

$$fitness = \sum((mean\_outer - (std\_outer/2)) - (mean\_inner + (std\_inner/2))) \quad \text{(B.26)}$$

$$fitness = \sum(mean\_outer - mean\_inner) \quad \text{(B.27)}$$

A partir dessas novas equações será possível analisar melhor o efeito dos desvios-padrões das distâncias interna e externa. O cálculo do *fitness* até então procurava atributos que mapeassem classes de forma densa, entretanto, o desvio-padrão subtraído da média de distâncias externas e o somado à média das distâncias internas pode atrapalhar a busca por classes mais separadas.

Observando-se novamente a Figura 49 mesmo a classe círculo sendo densa, o desvio-padrão de suas distâncias externas será bastante alto por causa da pouca densidade da classe triângulo, e da maior distância da classe losango. O desvio-padrão das distâncias

Tabela 48 – Desempenho do DCIAPSO.M8 e DCIAPSO.M9 utilizando as métricas MAUC e Geração de Protótipos.

Base	**M8 MAUC**	**M9 MAUC**	**M8 #Protótipos**	**M9 #Protótipos**
Abalone	0,6524±0,0834	0,8285±0,0194	88±101,1954	276,4±25,5402
Arrhythmia	0,7189±0,0319	0,8162±0,0256	26,2±3,1937	55,8±7,2595
Balance Scale	0,5554±0,0186	0,7945±0,0297	6±1,4142	10±4,5277
Car Evaluation	0,9300±0,0182	0,9325±0,0097	19±2,9155	18,8±10,2811
Contraceptive	0,5379±0,0196	0,6246±0,0143	4,6±0,5477	10±7,5829
Dermatology	0,9515±0,0172	0,9447±0,0097	12,6±5,9414	16±6,0415
E.coli	0,9181±0,0361	0,8920±0,0343	27,2±6,9785	32,2±5,9330
Gene	0,5262±0,0084	0,8255±0,0198	5,6±1,8166	24,8±9,0111
Glass	0,6708±0,0747	0,8487±0,0735	17,6±12,7789	24,6±7,2319
Hayes-Roth	0,5889±0,0555	0,8668±0,0432	6±1,4142	16,8±5,4037
Horse	0,6065±0,0486	0,7276±0,0198	5,2±1,0954	12,2±7,6942
Nursery	0,8250±0,0305	0,8213±0,0315	23,4±17,1843	33,2±20,1172
Page Blocks	0,5806±0,0313	0,8438±0,0123	8±2,5495	32,8±18,7403
Post Operative	0,5698±0,0497	0,6454±0,0395	10,2±6,1400	16,8±3,9623
Satimage	0,9125±0,0196	0,9231±0,0081	51,8±21,5801	75,6±10,7145
Soybean	0,9802±0,0114	0,9779±0,0083	25,6±3,2094	60,4±40,8081
Thyroid	0,6011±0,1514	0,8927±0,0155	7,2±5,5857	15,4±3,4351
Wine	0,9773±0,0175	0,9829±0,0143	6,6±1,5166	8±2,9155
Yeast	0,8066±0,0187	0,8421±0,0369	84,8±8,3187	77,4±13,6492
Zoo	0,9705±0,0088	0,9857±0,0213	12,8±5,0695	12,4±7,2664

internas das classes triângulo e losango, por sua vez, terão um valor alto. Se for selecionado, somente o atributo representado pela abscissa, o desvio-padrão das distâncias internas da classe losango será menor. Contudo, a classe triângulo ainda assim terá um desvio-padrão com alto valor, e a distância entre as classes círculo e losango também fará com que o desvio-padrão das distâncias externas permaneça com um valor alto.

A maior quantidade de classes e atributos, em conjunto com quantidades diferentes de instâncias em cada classe, tende a deixar o cenário muito mais complexo. Desta forma os valores de desvio-padrão podem ter efeitos maiores ou menores, dependendo das classes de cada base. Portanto, é possível que não somente este cálculo de *fitness* tenha sido mal concebido, como também esta forma de se calcular não seja adequado para bases com múltiplas classes desbalanceadas.

Os resultados obtidos com o DCIAPSO.M8 e DCIAPSO.M9 são apresentados na Tabela 48. Nas primeiras duas colunas são apresentados os desempenhos de acordo com a métrica MAUC, e nas duas colunas seguintes a quantidade de protótipos gerados.

As comparações estatísticas entre o DCIAPSO.M7 e as duas modificações correntes

deixam claro que a subtração e soma dos desvios-padrões causavam diminuição no desempenho do algoritmo. O desempenho do DCIAPSO.M9 em relação ao DCIAPSO.M8 revela que a melhor alternativa é levar em consideração apenas as médias de distâncias.

Em relação ao DCIAPSO, a oitava modificação produz um desempenho pior em dez das vinte e uma bases de dados, ao mesmo tempo em que gera uma maior quantidade de protótipos. Por outro lado, a nona modificação somente produz uma piora de desempenho em duas bases de dados, contudo com a mesma quantidade de protótipos gerados.

A complexidade da disposição das instâncias dificulta o cálculo de *fitness* baseado nas distâncias. A seleção de atributos levando em consideração um classificador, como o 1NN, até então se mostrou melhor. Além disso, o algoritmo de seleção ser do tipo *wrapper* já é bastante custoso. A necessidade de se calcular as distâncias, médias e desvios-padrões entre todas as instâncias para cada subconjunto de atributos aumenta ainda mais o custo computacional nesta parte do algoritmo. Portanto, não somente o SYMON deve ser descartado, mas também o cálculo do *fitness*, como nas Equações (B.23), (B.26) e (B.27).

B.13 DCIAPSO.M10

A primeira modificação conduzida no DCIAPSO substitui a métrica MAUC pela métrica AUCarea. A substituição aconteceu durante o processo de ajuste dos protótipos executado pelo PSO. A motivação da modificação se deu pelo fato desta segunda métrica ser mais sensível aos valores mais baixos de AUCs obtidos com pares de classes (YIJING et al., 2016). Logo, se espera que um ajuste de protótipos com uma métrica mais sensível possa resultar em uma melhor classificação.

Os resultados obtidos com o DCIAPSO.M1 mostraram que a utilização desta métrica durante o processo de ajuste dos protótipos não resultou em melhoria de desempenho. Exceto pelo desempenho em uma base, o DCIAPSO.M1 foi similar ao DCIAPSO.

Através de outras modificações, como o DCIAPSO.M2, foi possível observar que a utilização de um algoritmo de seleção de atributos pode ter como resultado melhoria de desempenho em algumas bases. A forma como a seleção de atributos é conduzida também usa um algoritmo de busca e uma classificação dos dados disponíveis. A motivação para o DCIAPSO.M10 é, portanto, verificar se a utilização da métrica AUCarea pode ter algum efeito no desempenho final ao substituir o MAUC durante o processo de seleção de atributos.

O DCIAPSO.M10 pode então ser visto como um ajuste direto no DCIAPSO.M2. A diferença entre esses algoritmos se dá no cálculo do *fitness* durante a execução do CSO, no processo de seleção de atributos. Enquanto no DCIAPSO.M2 a métrica utilizada para o *fitness* é o MAUC, no DCIAPSO.M10 a métrica passa a ser o AUCarea.

Os resultados obtidos são apresentados na Tabela 49. Em comparação com o DCIAPSO, esta modificação teve melhoria de desempenho nas bases *Gene* e *Thyroid*, e de-

Tabela 49 – Desempenho do DCIAPSO.M10 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8342±0,0065	267,4±36,2188	0,9199
Arrhythmia	0,7885±0,0319	55,6±4,9800	0,8462
Balance Scale	0,7970±0,0178	9,2±3,9623	0,9816
Car Evaluation	0,9450±0,0177	34,2±8,0747	0,9753
Contraceptive	0,6216±0,0109	15,2±6,3008	0,9871
Dermatology	0,9754±0,0059	15,2±4,4385	0,9481
E.coli	0,9072±0,0302	26,2±6,4187	0,9025
Gene	0,8243±0,0295	31,2±6,9426	0,9878
Glass	0,8654±0,0511	32,2±15,4337	0,8119
Hayes-Roth	0,8653±0,0533	12,2±7,1204	0,9047
Horse	0,7231±0,0484	10,4±6,6558	0,9645
Nursery	0,8181±0,0256	19±10,7703	0,9982
Page Blocks	0,8615±0,0127	27,2±8,8713	0,9938
Post Operative	0,6296±0,0878	18,2±6,2209	0,7472
Satimage	0,9250±0,0115	73,2±14,8223	0,9858
Shuttle	0,9778±0,0042	47,4±10,3827	0,9990
Soybean	0,9718±0,0105	46,4±25,1257	0,9079
Thyroid	0,9734±0,0066	16±6,4031	0,9972
Wine	0,9806±0,0071	6,4±2,8810	0,9551
Yeast	0,8114±0,0311	60,4±24,1930	0,9491
Zoo	0,9594±0,0149	16,2±11,4324	0,7995

sempenho estatisticamente equivalente no restante das bases. Ambos os casos de melhoria aconteceram em bases com a presença de uma classe majoritária que possui mais instâncias que as duas classes minoritárias juntas.

Uma comparação feita com o DCIAPSO.M2 mostra que ambos os algoritmos possuem desempenhos estatisticamente similares em praticamente todas as bases. Na base *Wine* o DCIAPSO.M10 produz um desempenho melhor. Esta base é uma das que mais sofre modificações nos valores de seus atributos. Neste caso, a normalização dos valores seguido de uma seleção de atributos produziu melhores resultados com a utilização de uma métrica mais sensível.

B.14 DCIAPSO.M11

O sucesso obtido com o DCIAPSO.M10 serviu como motivação para que um teste similar fosse realizado com o DCIAPSO.M4. Enquanto no DCIAPSO.M2 a seleção de atributos foi realizada com CSO, na quarta modificação a seleção de atributos foi realizada

Tabela 50 – Desempenho do DCIAPSO.M11 utilizando as métricas MAUC, Geração e Redução de Instâncias.

Base	**MAUC**	**#Protótipos**	**Redução**
Abalone	0,8301±0,0156	256,6±26,4821	0,9231
Arrhythmia	0,8127±0,0509	57±8,6313	0,8424
Balance Scale	0,7930±0,0159	12,2±2,1679	0,9756
Car Evaluation	0,9403±0,0246	31,6±6,2690	0,9771
Contraceptive	0,6491±0,0180	12,6±3,6469	0,9893
Dermatology	0,9653±0,0154	21,8±8,1670	0,9255
E.coli	0,8872±0,0213	27,4±3,7815	0,8981
Gene	0,7638±0,0184	52,6±5,4129	0,9794
Glass	0,8640±0,0389	32±5,5678	0,8131
Hayes-Roth	0,8764±0,0566	14±3,6742	0,8906
Horse	0,7192±0,0820	20,6±5,3198	0,9296
Nursery	0,8452±0,0333	18,2±7,1903	0,9982
Page Blocks	0,8457±0,0395	33,2±11,6919	0,9924
Post Operative	0,6169±0,0842	17,4±4,4497	0,7583
Satimage	0,9240±0,0053	67±8,2158	0,9870
Soybean	0,9756±0,0137	51±16,8967	0,8988
Thyroid	0,9552±0,0165	13,4±3,9749	0,9977
Wine	0,9795±0,0184	6,6±3,6469	0,9537
Yeast	0,8448±0,0393	80,4±12,2188	0,9323
Zoo	0,9709±0,0272	19±16,8226	0,7649

com WOA-CM. O cálculo do *fitness* da segunda modificação sendo feito com a métrica AUCarea resultou em melhoria de desempenho. Desta forma, a modificação corrente foi implementada para se verificar se há melhoria de desempenho caso a métrica AUCarea seja também utilizada durante a seleção de atributos com o WOA-CM.

Os resultados deste experimento podem ser consultados na Tabela 50. Ao ser comparado ao algoritmo DCIAPSO esta modificação se mostra pior na base *E.coli* e melhor na base *Thyroid*. A utilização do WOA-CM já permitia ao DCIAPSO uma melhora de desempenho nesta segunda base. A utilização da métrica AUCarea, entretanto, prejudicou a seleção de atributos na primeira base citada.

Ao ser comparado com o DCIAPSO.M10, a modificação corrente se mostra pior apenas na base *Gene*. Em relação à base *E.coli*, o desempenho do DCIAPSO.M10 pode ser visto como um "meio-termo" entre o DCIAPSO e o DCIAPSO.M11. Desta forma, é possível que a décima modificação seja estatisticamente equivalente a ambos, mesmo o DCIAPSO sendo estatisticamente superior à décima-primeira modificação.