CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Diêgo Gomes Piovezana

AVALIAÇÃO DE RISCO DE INADIMPLÊNCIA DE USUÁRIOS DE CARTÃO DE CRÉDITO UTILIZANDO APRENDIZADO DE MÁQUINA - ESTUDO DE CASO

Timóteo

2022

Diêgo Gomes Piovezana

AVALIAÇÃO DE RISCO DE INADIMPLÊNCIA DE USUÁRIOS DE CARTÃO DE CRÉDITO UTILIZANDO APRENDIZADO DE MÁQUINA - ESTUDO DE CASO

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Deisymar Botega Tavares
Coorientadora: Andressa Oliveira Souza

Timóteo

2022

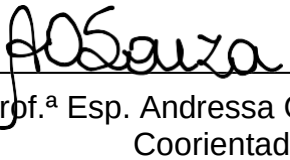
Diêgo Gomes Piovezana

AVALIAÇÃO DE RISCO DE INADIMPLÊNCIA DE USUÁRIOS DE CARTÃO DE CRÉDITO UTILIZANDO APRENDIZADO DE MÁQUINA - ESTUDO DE CASO

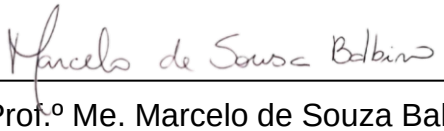
Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 15 de março de 2023:

Prof.ª Me. Deisymar Botega Tavares
Orientadora



Prof.ª Esp. Andressa Oliveira Souza
Coorientadora



Prof.º Me. Marcelo de Souza Balbino
Professor Convidado

Timóteo
2023

Agradecimentos

É com satisfação que finalizo a elaboração desta monografia, como consequência de anos de dedicação e estudos durante minha graduação. Gostaria de expressar minha gratidão a todos aqueles que contribuíram de alguma forma para a realização deste trabalho.

Agradeço pela paciência, sabedoria e disponibilidade dos professores que me motivaram a superar cada desafio enfrentado ao longo desta jornada acadêmica. Expresso minha gratidão a todos os profissionais e também amigos do CEFET-MG. Agradeço aos colegas de curso, que compartilharam seus conhecimentos e experiências, tornando esta trajetória ainda mais enriquecedora e agradável.

De modo especial, agradeço à Andressa por todo precioso conhecimento, suporte e encorajamento fornecido durante todo o processo de pesquisa, implementação e escrita deste trabalho. Expresso minha gratidão à Deisymar pela excelente orientação, por toda a dedicação, paciência e orientações compartilhadas ao longo deste processo, sobretudo no que diz respeito a escrita da minha monografia. Este trabalho coletivo certamente contribuiu significativamente para o êxito desta monografia. Parabenizo pelas excelentes profissionais e pessoas que são, pois foram verdadeiras mentoras e exemplos a serem seguidos.

Por fim, mas não menos importante, agradeço à Deus, a todos meus amigos e à minha família, que, direta ou indiretamente sempre me apoiaram e incentivaram a seguir meus objetivos, pelo amor, carinho e compreensão em todos os momentos. Expresso minha profunda gratidão a todos os que ajudaram a tornar este trabalho possível. Espero que meu trabalho possa de alguma maneira contribuir para a comunidade científica e acadêmica.

“Os grandes navegadores
devem sua reputação aos temporais e tempestades”.
Epicuro

Resumo

A avaliação de risco de inadimplência de clientes de cartão de crédito é uma tarefa importante para assegurar uma boa saúde financeira das organizações. Realizar uma análise de risco de crédito de forma eficiente é um desafio para as instituições financeiras, pois a quantidade de dados disponíveis é muito grande e a análise de risco de crédito é uma tarefa complexa. Isso leva a uma necessidade de ferramentas e técnicas que sejam capazes de realizar análises eficazes e eficientes a fim de auxiliar no planejamento estratégico dessas instituições. Assim sendo, este trabalho apresenta uma abordagem para a previsão de inadimplência de usuários de cartão de crédito, utilizando algoritmos classificadores e técnicas de balanceamento de dados. A abordagem proposta é baseada em um modelo de aprendizado de máquina supervisionado, que utiliza um conjunto de dados de clientes de cartão de crédito, com informações sobre o histórico de pagamentos e outras informações sobre o cliente. O conjunto de dados utilizado possui cerca de 5,5 milhões de registros anônimos e é proveniente plataforma Kaggle, através de uma competição AMEX intitulada "*AmExpert 2021 CODELAB – Machine Learning Hackathon*". Os classificadores utilizados foram selecionados a partir de uma análise de literatura, sendo eles: *XGBoost*, *Random Forest* e *Logistic Regression*. Os resultados obtidos foram satisfatórios, pois os classificadores tiveram seus resultados comparados e *Random Forest*, com acurácia de 89,92%, foi capaz de classificar os usuários de cartão de crédito inadimplentes presentes em uma base balanceada. Além disso, obteve sensibilidade de 93,68%, especificidade de 86,16%, *FPR Fraud* de 13,84%, precisão de 87,13% e *F-Score* de 90,28%. Assim sendo, este trabalho foi capaz de prover uma compreensão sobre a utilização de algoritmos classificadores a fim de realizar a previsão de inadimplência através da análise de uma base de dados de usuários de cartão de crédito de modo a representar uma importante ferramenta podendo ser utilizada aliada aos analistas de crédito provendo melhorias na qualidade de seus trabalhos.

Palavras-chave: análise de crédito, aprendizado de máquina, cartão de crédito, inadimplência.

Abstract

Assessing the risk of default on credit card customers is an important task to ensure good financial health for organizations. Carrying out an efficient credit risk analysis is a challenge for financial institutions, as the amount of data available is enormous and credit risk analysis is a complex task. This leads to a need for tools and techniques that be able to carry out effective and efficient analyzes in order to assist in the strategic planning of these institutions. Therefore, this paper presents an approach for predicting bad debt for credit card users, using classifier algorithms and data balancing techniques. The proposed approach is based on a supervised machine learning model, which uses a dataset of credit card customers, with information about payment history and other customer information. The dataset used has about 5.5 million anonymous records and comes from the Kaggle platform, through an AMEX competition entitled "*AmExpert 2021 CODELAB – Machine Learning Hackathon*". The classifiers used were selected from a literature review, namely: *XGBoost*, *Random Forest* and *Logistic Regression*. The results obtained were satisfactory, as the classifiers had their results compared and *Random Forest*, with an accuracy of 89.92% , was able to classify delinquent credit card users present on a balanced basis. In addition, it obtained sensitivity of 93.68%, specificity of 86.16%, *FPR Fraud* of 13.84%, accuracy of 87.13 % and *F-Score* of 90.28%. Therefore, this work was able to provide an understanding of the use of classifier algorithms in order to predict default default risk, by analyzing a credit card user database, to represent an important tool that can be used allied to credit analysts providing improvements in the quality of their work.

Keywords: credit analysis, machine learning, credit card, default.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Abecs	Associação Brasileira das Empresas de Cartões de Crédito e Serviços
AUC	*Area Under the Curve*
CART	Espaço de árvores de regressão
GBM	Algoritmo *Gradient Boosting Machine*
GBRT	Algoritmo *Gradient Boosting Regression Tree*
KNN	Algoritmo *K-Nearest Neighbors*
KS	*Kolmogorov-Smirnov*
ROC	*Receiver Operating Characteristic*
SMOTE	*Synthetic Minority Oversampling Technique*
SVM	Algoritmo *Support Vector Machine*
RF	Algoritmo *Random Forest*
LR	Algoritmo *Logistic Regression*
XGB	Algoritmo *Extreme Gradient Boosting*

Lista de símbolos

δ	Letra grega delta
Γ	Letra grega gama
$\in$	Símbolo de pertence
$\mathbb{R}$	Conjunto dos números reais
Ω	Letra grega ômega
ϕ	Letra grega phi
$\sum$	Símbolo de somatório
Θ	Letra grega theta
$\rightarrow$	Símbolo matemático de implicação
ρ	Letra grega rho
σ	Letra grega sigma

Sumário

1 Introdução

O uso do cartão de crédito está cada vez mais presente na vida de muitos brasileiros. De acordo com uma pesquisa realizada pela Associação Brasileira das Empresas de Cartões de Crédito e Serviços (Abecs), os pagamentos com cartões de crédito cresceram 42,4% no primeiro trimestre de 2022 em comparação com o período de janeiro a março de 2021, movimentando R$ 478,5 bilhões nos três primeiros meses do ano (MELLO, 2022). Segundo pesquisa de Endividamento e Inadimplência do Consumidor divulgada em março de 2022, pela Confederação Nacional do Comércio de Bens, Serviços e Turismo, o percentual de famílias que relataram ter dívidas a vencer alcançou 77,5%, a maior proporção já registrada nos 12 anos do levantamento, com o cartão de crédito representando 87,0% do total de famílias endividadas no país (NETO, 2022).

Como tentativa de solução para o problema de inadimplência, a prevenção pode ser uma estratégia mais eficaz se comparado à remediação. Quando um indivíduo ou empresa não consegue realizar o pagamento da fatura de cartão de crédito, pode ser criado um efeito cascata. Afinal, fica cada vez mais difícil levantar o dinheiro para quitar o que se deve, e é possível que o consumidor comece a atrasar outras contas, gerando um ciclo difícil de ser quebrado (MENASCE, 2021). Esta dificuldade que um cliente apresenta para quitar suas dívidas, e consequentemente, não devolver o dinheiro cedido pelo banco através do limite do cartão no tempo correto, não se trata apenas de um problema para o cliente, que terá sua vida financeira prejudicada, mas também para o banco, que não irá receber o seu dinheiro.

Compreender o risco de inadimplência de uma pessoa física ou jurídica é fundamental para a existência de uma saúde financeira em bom estado. Segundo anúncio realizado por um dos maiores bancos digitais brasileiros, em maio de 2022, o Nubank apresentou uma taxa de inadimplência de 4,2% e prejuízo líquido de quase US$ 45 milhões (cerca de 222 bilhões de reais) e diante disso, o banco reforçou a previsão para devedores duvidosos para US$ 921 milhões (aproximadamente 4,5 bilhões de reais) nos primeiros meses de 2022 (JUNIOR, 2022).

Os analistas de crédito são profissionais que classificam o risco de crédito de cada tomador do empréstimo, ou seja, a sua capacidade de honrar o empréstimo concedido. É a partir da sua análise, que serão tomadas as decisões relativas à aplicação de valiosos recursos que, se bem orientados gerarão retornos positivos à empresa (CRUZ, 2018). Quando os analistas de crédito possuem à sua disposição uma regra de reconhecimento de padrões e classificação que indique previamente a chance de inadimplência de um futuro cliente, a decisão de concessão de crédito é facilitada; esse profissional pode então utilizar argumentos quantitativos em substituição a argumentos subjetivos e decidir com maior confiança (GUIMARÃES; NETO, 2002).

Segundo Fernandes (2019), o volume de dados a ser gerenciado pelas empresas tem se elevado de tal forma que o tratamento de grandes bases de dados supera a habilidade hu-

mana de entender e de eficientemente lidar com esses dados. O que leva a uma necessidade crítica de ferramentas e técnicas que sejam capazes de realizar automaticamente análises eficazes e eficientes dos dados para apoiar empresas e indivíduos no planejamento estratégico. Assim, considerando este cenário, utilizar técnicas de aprendizado de máquina pode ser uma estratégia pertinente em prol da predição de inadimplências e prevenção a endividamentos, superior a habilidade humana, visto que a inteligência artificial é capaz de produzir, rápido e automaticamente, modelos capazes de analisar dados maiores e mais complexos, e entregar resultados mais rápidos e precisos – mesmo em grande escala (SAS, 2022).

Destarte, este trabalho busca responder a seguinte questão de pesquisa: é possível prever a inadimplência de usuários de cartão de crédito presentes a partir de uma base de dados pública, através da utilização de algoritmos classificadores?

1.1 Justificativa

O mau uso do crédito pode afetar a qualidade de vida dos cidadãos, levar ao endividamento massivo da população e também da cadeia produtiva, e impactar inclusive a macroeconomia (RIBEIRO; LARA, 2016).

A inadimplência pode causar sérios transtornos para instituições financeiras, que podem ser impactadas com o aumento do número de clientes inadimplentes, pois isso expõe os bancos a maiores riscos na concessão de empréstimos, necessitando limitar as possibilidades aos maus pagadores, impossibilitando o acesso ao crédito por meio de mais garantias ou informações sobre o tomador de crédito e elevando os juros. Com os juros mais altos, menos pessoas vão se sentir propensas a tomar empréstimo ou a continuar pagando a dívida contratada (MENASCE, 2021). Além disso, em uma empresa de crédito, a inadimplência afeta o desenvolvimento, que consequentemente reduz a receita e aumenta os gastos com os processos de cobrança (PAIXÃO; TEIXEIRA, 2019). Para a economia, de modo geral, o nível de inadimplência da população é sempre um fator preocupante. Isso, porque, se muitas pessoas deixam de pagar as suas dívidas, as empresas passam por dificuldades de funcionamento (MENASCE, 2021).

Segundo Santos (2007), para minimizar o risco de crédito, destaca-se cada vez mais a importância da gestão do risco de crédito, baseada em procedimentos subjetivos (análise caso a caso) e objetivos (análise estatística), como instrumento para a adequada seleção, análise, precificação e, principalmente, monitoramento do risco de inadimplência, quando da ocorrência de fatores sistêmicos adversos.

A decisão de oferecer ou não crédito a um cliente deve ser bastante criteriosa. Dar crédito ao cliente em perspectiva é determinado pelas suas características relacionadas, como idade, renda, solvência e escolaridade e de características do crédito como tipo de crédito, vencimento, valor do empréstimo e outras características inerentes às operações financeiras (CHEN; HUANG, 2003).

O aprendizado de máquina é uma área da inteligência artificial cujo objetivo é desenvolver técnicas computacionais e de construções de sistemas capazes de adquirir conhecimento

de forma automática e utilizável (FONSECA; NETO; SOUZA, 2005). Segundo estudo publicado por Aniceto (2016), ao observar a utilização dos classificadores como Decision Tree, Random Forest, Support Vector Machines, Bagging e AdaBoost, foi possível inferir que o aprendizado de máquina tem sido bastante utilizado na avaliação do risco de crédito pois apresenta resultados competitivos com os métodos tradicionalmente utilizados. Muitos pesquisadores têm obtido resultados promissores sobre previsão de notação de crédito utilizando diferentes técnicas estatísticas e Inteligência Artificial (HUANG et al., 2004).

Em consonância a esse raciocínio, técnicas de aprendizado de máquina podem ajudar a extrair relações dos dados históricos financeiros de usuários de cartão de crédito e, consequentemente, assegurar decisões mais assertivas, podendo facilmente analisar uma quantidade massiva de informações, sobretudo se comparado à capacidade humana. Gerentes de crédito podem ser auxiliados a aceitar ou negar a realização de empréstimos, auxiliando as empresas financeiras na previsão de possíveis inadimplências de seus clientes. Assim, ao utilizar destas técnicas, haveria menos risco para os bancos, por exemplo, ao prever quais usuários de cartão de crédito serão bem-sucedidos em seus pagamentos.

Dessa forma, faz-se necessária uma avaliação automática, utilizando técnicas rápidas e adaptativas, onde a probabilidade de inadimplência pode ser calculada a partir de um conjunto de dados históricos em um período de tempo razoável (FERNANDES, 2019). O que vai ao encontro do objetivo deste trabalho, por meio da aprendizagem de máquina.

1.2 Objetivos

O presente trabalho possui como objetivo geral utilizar diferentes técnicas de aprendizado de máquina aplicadas a um conjunto de dados contendo o perfil agregado de diversos clientes para, com base nas características presentes, prever se um cliente será ou não capaz de quitar integralmente sua fatura de cartão de crédito dentro do prazo e manter-se adimplente.

Entre os objetivos específicos estão:

1. Verificar se o uso do algoritmo de sobreamostragem SMOTE contribui para a previsão de clientes inadimplentes;
2. Analisar o comportamento dos classificadores quando submetidos a versões alternativas da mesma base de dados construídas de diferentes maneiras;
3. Comparar os resultados a partir de distintos algoritmos de classificação para a base de dados selecionada, a fim de considerar o mais eficiente neste contexto.

2 Fundamentação Teórica

Com o objetivo de promover uma melhor compreensão ao leitor, neste capítulo são apresentados os principais conceitos relacionados ao desenvolvimento deste trabalho.

2.1 O aprendizado de máquina

Surgido como campo da inteligência artificial na década de 60, o aprendizado de máquina possuía como objetivo aprender padrões com base em dados e possuíam cunho estritamente computacional. Porém, desde então esta área expandiu-se de modo a interceder-se com áreas da estatística (IZBICKI; SANTOS, 2020).

A aprendizagem de máquina possui como finalidade criar técnicas computacionais sobre aprendizado e desenvolvimento de sistemas inteligentes, com capacidade para tomar decisões através de experiências acumuladas (HENDRICKX et al., 2015 apud GANDRA, 2020). É capaz de se modificar de maneira automática de modo a tornar-se mais eficiente cada vez que realiza a mesma tarefa sobre um mesmo domínio de conhecimento (SANCHES, 2003 apud GANDRA, 2020).

Técnicas baseadas em aprendizado de máquina têm sido aplicadas com sucesso em diversas áreas que vão desde reconhecimento de padrões, engenharia de espaçonaves, visão computacional, entretenimento, biologia computacional, finanças e até aplicações biomédicas e médicas (NAQA; MURPHY, 2015).

O aprendizado de máquina consiste em técnicas de aprendizado supervisionado, não-supervisionado, ou semi-supervisionado (FERNANDES, 2019). Cada tipo de aprendizado será detalhado a seguir.

2.1.1 Aprendizado supervisionado

Redes de aprendizado supervisionado são capazes de aprender ao serem apresentadas a dados de treinamento pré-classificados (COPPIN, 2017 apud ANDRADE, 2019). Os algoritmos de aprendizagem supervisionada relacionam uma saída com uma entrada baseando-se em dados rotulados (FONTANA, 2020).

Caso o tipo da classe seja contínuo, o problema de indução é chamado de regressão (SANCHES, 2003). Caso seja discreto, ou seja, quando a saída pode assumir um conjunto de rótulos pré-definidos (FONTANA, 2020), o problema é chamado de classificação (SANCHES, 2003).

A aprendizagem supervisionada possui o objetivo de construir um modelo capaz de aprender o mapeamento entre valores das variáveis independentes e o valor da variável dependente das instâncias contidas em um conjunto de treinamento tal que o classificador resultante possua um poder de generalização suficiente para ser utilizado para predição de instân-

cias nunca antes vistas (KOTSIANTIS et al., 2007 apud SCHNEIDER, 2018).

Alguns exemplos de algoritmos são: *Random Forest*, *Naive Bayes*, *K-nearest neighbors* (KNN), *Support vector machine* (SVM), *Decision tree*, *Logistic Regression*, entre outros (SINGH; HALGAMUGE; LAKSHMIGANTHAN, 2017; EL-MAGD; ALI; PHAM, 2021).

2.1.2 Aprendizado não supervisionado

No aprendizado não supervisionado, não é atribuído um rótulo para os dados de saída e o algoritmo busca padrões e similaridades entre os dados, permitindo identificar grupos de itens similares ou similaridade de itens novos com grupos já definidos (FONTANA, 2020).

Os algoritmos não supervisionados são geneticamente divididos em: agrupamento (na qual os dados são agrupados de acordo com sua similaridade), sumarização (cujo objetivo é encontrar uma descrição simples e compacta para um conjunto de dados) e associação, que consiste em encontrar padrões de associações entre atributos de um conjunto de dados (FACELI et al., 2011 apud ANDRADE, 2019).

Em alguns casos, pode ser aplicado um conjunto de dados parcialmente rotulados que tende a melhorar significativamente o desempenho dos algoritmos de aprendizagem não supervisionada. Esta abordagem hibrida é normalmente chamada de aprendizagem semi-supervisionada (FONTANA, 2020).

Um exemplo de algoritmo não supervisionado é o *K-means* (BHIMANI; LEESER; MI, 2015).

2.1.3 Aprendizado por reforço

O aprendizado por reforço é um campo do aprendizado de máquina com foco na criação de agentes capazes de tomar decisões acertadas em um ambiente sem que se tenha qualquer conhecimento prévio sobre este ambiente. Para que ocorra aprendizado é preciso que o agente após perceber o ambiente tome ações e observe a recompensa imediata proveniente da ação e a mudança no ambiente decorrente da ação tomada (TEIXEIRA, 2016).

O objetivo do aprendizado por reforço é o desenvolvimento de um agente capaz de tomar decisões e interagir com seu ambiente com base em suas observações do ambiente de modo a maximizar o total de recompensas imediatas recebidas durante a interação com o ambiente. A escolha da decisão a ser tomada a cada instante é feita de forma a garantir a maximização de uma certa função objetivo, que atribui recompensa a estados favoráveis para o agente. Desta forma, o agente não está sempre interessado em ações que geram a maior recompensa imediata, mas sim em ações que o leve ao maior acúmulo de recompensa a longo prazo (TEIXEIRA, 2016).

2.2 Algoritmos classificadores

Os algoritmos de classificação são algoritmos de aprendizagem de máquina supervisionada na qual o objetivo é realizar a previsão do rótulo (ou classe) associado a uma variável

de entrada contendo determinados atributos (FONTANA, 2020).

2.2.1 XGBoost

O *XGBoost* ou *Extreme Gradient Boosting* foi desenvolvido por Chen e Guestrin (2016) e é um algoritmo de aprendizado de máquina capaz de lidar com dados esparsos e utilizando um esboço de quantil ponderado teoricamente justificado para aprendizado aproximado e cálculo eficiente da proposta, realizando padronizações de acesso ao cache, compressão e fragmentação de dados para construir um sistema de *Gradient Boosting* (também conhecido como *Gradient Tree Boosting*, *Gradient Boosting Machine (GBM)* ou *Gradient Boosted Regression Tree (GBRT)*) de ponta a ponta altamente escalável utilizando poucos recursos computacionais e reconhecido em diversos desafios de aprendizado de máquina e mineração de dados (CHEN; GUESTRIN, 2016).

Segundo seus autores, o algoritmo do *XGBoost* é capaz de fornecer resultados de última geração em uma ampla gama de problemas, com escalabilidade em todos os cenários. Chen e Guestrin (2016) afirmam que a computação paralela e distribuída utilizada pelo algoritmo torna o aprendizado mais rápido, o que permite uma exploração mais rápida do modelo. Além disso, a estrutura de bloco com reconhecimento de cache proposto por eles, é eficaz para um aprendizado *out-of-core* (em português, fora do núcleo do processador).

Em um conjunto de dados com n exemplos e m características $\mathcal{D} = (x_i, y_i)$ ($|\mathcal{D}| = n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}$) um modelo de conjunto de árvore utiliza K funções aditivas para prever a saída.

$$\widehat{y} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i),\ f_k \in \mathcal{F} \tag{2.1}$$

Na Equação 2.1 onde $\mathcal{F} = \{f(x) = w_q(x)\}(q : \mathbb{R}^m \to T, w \in \mathbb{R}^T)$ é o espaço de árvores de regressão (conhecida também como CART). Nesta equação, q representa a estrutura de cada árvore que mapeia um exemplo para o índice de folha correspondente. T é a quantidade de folhas na árvore. Cada f_k corresponde a uma estrutura de árvore independente q e pesos de folha w. Diferentemente das *Decision Trees*, cada árvore de regressão contém uma pontuação contínua em cada folha. w_i é usado para representa a pontuação da i-ésima folha. Como exemplo, as regras de decisão foram usadas nas árvores (representadas por q) para classificá-las nas folhas e calcular a previsão final somando a pontuação nas folhas correspondentes (dadas por w) (CHEN; GUESTRIN, 2016).

Para aprender o conjunto de funções utilizadas no modelo, foi minimizado o objetivo regularizado representado pela Equação 2.2 abaixo:

$$\mathcal{L}(\phi) = \sum_{1} l(\widehat{y}_i, y_i) + \sum_{k} \Omega(f_k) \tag{2.2}$$

$$\text{onde } \Omega(f) = \gamma T + (1/2)\gamma||w||^2$$

Nesta equação, l é uma função de perda convexa diferenciável que mede a diferença entre a previsão $\widehat{y}_i$ e o alvo y_i. O segundo termo Ω penaliza a complexidade do modelo (as funções da árvore de regressão). O termo de regularização adicional ajuda a suavizar os pesos finais aprendido para evitar um ajuste excessivo. Dessa forma, de maneira intuitiva o objetivo regularizado tenderá a selecionar um modelo empregando funções simples e preditivas. Neste cenário, caso o parâmetro de regularização seja definido como zero, o objetivo retorna para a tradicional *gradient tree boosting* (CHEN; GUESTRIN, 2016).

Devido ao fato do modelo de conjunto de árvores na Equação 2.2 incluir funções como parâmetros e não poder ser otimizado utilizando métodos tradicionais de otimização no espaço euclidiano, o modelo é treinado de maneira aditiva. Formalmente temos $\widehat{y}_i^{(t)}$ representando a previsão da i-ésima instância na t-ésima iteração, precisa-se adicionar f_t para minimizar o objetivo representado pela Equação 2.3:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \widehat{y}_i^{(t-i)} + f_t(x_i)) + \Omega(f_t) \tag{2.3}$$

Isto significa que será adicionado o f_t que mais trouxer melhoria para o modelo de acordo com a Equação 2.2 (CHEN; GUESTRIN, 2016). A aproximação de segunda ordem pode ser usada para otimizar rapidamente o objetivo no cenário geral (FRIEDMAN; HASTIE; TIBSHIRANI, 2000 apud CHEN; GUESTRIN, 2016). Desse modo, temos a Equação 2.4:

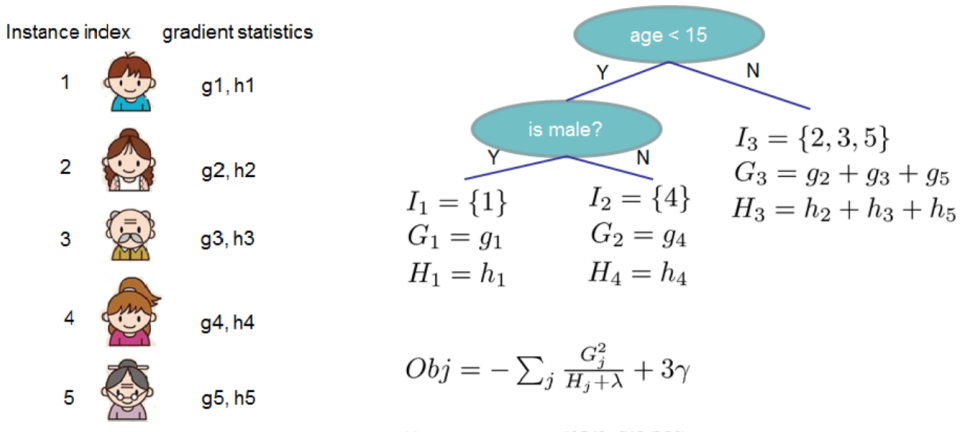
$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \widehat{y}_i^{(t-i)} + g_i f_t(x_i) + (1/2) h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.4}$$

onde $g_i = \delta_{\widehat{y}^{(t-1)}} l(y_i, \widehat{y}^{(t-1)})$ e $h_i = \delta^2_{\widehat{y}^{(t-1)}} l(y_i, \widehat{y}^{(t-1)})$ são estatísticas de gradiente de primeira e segunda ordem na função de perda. Removendo os termos constantes, obtém-se a Equação 2.5 como objetivo simplificado no passo t:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + (1/2) h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.5}$$

Para calcular a pontuação da estrutura, é necessário apenas somar o gradiente e as estatísticas do gradiente de segunda ordem em cada folha e então aplicar a fórmula da pontuação para obter o índice de qualidade, conforme pode ser observado na Figura 1. Quanto menor a pontuação, melhor é a estrutura (CHEN; GUESTRIN, 2016).

Figura 1 – Cálculo da pontuação da estrutura



Fonte: (CHEN; GUESTRIN, 2016)

Sendo $I_j = \{i|q(x_i)\} = j$ como o conjunto de instâncias da folha j. Podemos reescrever a Equação 2.5 expandindo Ω da seguinte forma:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\gamma\sum_{i=1}^{n} w_j^2 \tag{2.6}$$

$$= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \gamma)w_j^2\right] + \gamma T \tag{2.7}$$

Para uma estrutura fixa $q(x)$ pode-se calcular o peso ótimo w_j^* para cada folha j usando a Equação 2.8:

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{2.8}$$

e calcular o valor ideal correspondente por:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\left[\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j h_i+\lambda}} +\right] + \gamma T \tag{2.9}$$

A Equação 2.9 pode ser usadas como uma função de pontuação para avaliar a qualidade de uma estrutura q (CHEN; GUESTRIN, 2016). Essa pontuação é como a pontuação de impureza para avaliar árvores de decisão, exceto pelo fato que é derivada para uma gama mais ampla de funções objetivo. A Figura 1 ilustra como esta pontuação pode ser calculada.

Normalmente é impossível enumerar todas as estruturas de árvores possíveis q. Ao invés disso, é usado um algoritmo guloso que inicia a partir de uma única folha e adiciona iterativamente ramos à árvore. Suponha que I_L e I_R são os conjuntos de instâncias dos nós esquerdo e direito após a divisão (CHEN; GUESTRIN, 2016). Deixando $I = I_L \cup I_R$, então a redução da perda após a divisão é dada por:

$$\mathcal{L}_{\text{divisão}} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{2.10}$$

A Equação 2.10 é geralmente usada na prática para avaliar os candidatos divididos.

Duas técnicas adicionais são utilizadas para evitar ainda mais o ajuste excessivo (*overfitting*). Introduzido por Friedman (2002), a primeira técnica é a retração (*shrinkage*), que dimensiona os pesos recém adicionados por um fator η após cada etapa do aumento da árvore. Semelhante a uma taxa de aprendizado na otimização estocástica, a retração reduz a influência de cada árvore individual e deixa espaço para futuras árvores melhorarem o modelo. A segunda técnica é a subamostragem de coluna (*column subsampling*), que evita o sobreajuste ainda mais ao se comparado a subamostragem de linha tradicional (CHEN; GUESTRIN, 2016).

Um dos principais problemas no aprendizado de árvore é encontrar a melhor divisão conforme a Equação 2.10. Para resolver esse problema, um algoritmo de localização de divisão enumera todas as divisões possíveis em todos os atributos, o que pode ser chamado de *exact greedy algorithm*, ou em tradução livre, algoritmo ganancioso exato. A Figura 2 apresenta o algoritmo, onde a entrada I é o conjunto de instâncias do nó atual, a entrada d é a dimensão e como saída está o *score* (pontuação) máximo de divisão.

Figura 2 – Algoritmo ganancioso exato para descoberta de divisão

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i\in I} g_i$, $H \leftarrow \sum_{i\in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Fonte: (CHEN; GUESTRIN, 2016)

No entanto, esse algoritmo é muito custoso computacionalmente pois enumera todas as divisões possíveis para atributos contínuos. Para fazer isso de forma eficiente, o algoritmo deve primeiro classificar os dados de acordo com os valores dos atributos e visitar os dados de maneira ordenada para acumular estatísticas de gradiente para a pontuação de estrutura conforme Equação 2.10.

O algoritmo guloso exato é muito poderoso, pois enumera de forma gananciosa todos os possíveis pontos de divisão. No entanto, é impossível fazê-lo com eficiência quando os dados não cabem inteiramente na memória.

O algoritmo apresentado na Figura 2 propõe que se pode ditar pontos de divisão de acordo com percentis de distribuição de características. Então mapeia os atributos contínuos em *buckets* divididos por esses pontos candidatos, agrega as estatísticas e encontra a melhor solução entre as propostas com base nas estatísticas agregadas (CHEN; GUESTRIN, 2016).

Há duas variantes do algoritmo, dependendo de quando a propostas é apresentada. A variante global propõe todas as divisões candidatas durante a fase inicial da construção da árvore, e usa as mesmas propostas para encontrar a divisão em todos os níveis. A variante local é reproposta após cada divisão. O método global requer menos etapas de proposta do que o método local. No entanto, geralmente mais pontos candidatos são necessários para a proposta global porque os candidatos não são refinados após cada divisão. A proposta local refina os candidatos após as divisões e pode ser mais apropriada para árvores mais profundas. Assim, a proposta local de fato requer menos candidatos e a proposta global pode ser tão precisa quanto a local se houver candidatos suficientes (CHEN; GUESTRIN, 2016).

Um passo importante no algoritmo aproximado é propor pontos de divisão candidatos. Normalmente percentis de um atributo são usados para fazer com que os candidatos distribuam uniformemente. Formalmente, o multiconjunto $\mathcal{D}_k = \{(x_{1k}, h_1), (x_{2k}, h_2)...(x_{nk}, h_n)\}$ representa os k-ésimos valores de atributo e estatísticas de gradiente de segunda ordem de cada instância de treinamento. Podemos definir uma função $r_k : \mathbb{R} \to [0, +\infty]$ como:

$$r_k(z) = \frac{1}{\sum(x,h) \in \mathcal{D}_k} \sum_{(x,h)\in\mathcal{D}_k, x<z} h, \tag{2.11}$$

que representa a proporção de instâncias cujo valor de característica k é menor que z. O objetivo é encontrar pontos de divisão candidatos $s_{k1}, s_{k2}, ..., s_{kl}$, de modo que

$$|r_k(s_{ki}) - r_k(s_{kj})| \leq \epsilon, s_{k1} = min_i x_{ik}, s_{kl} = max_i x_{ik}, \tag{2.12}$$

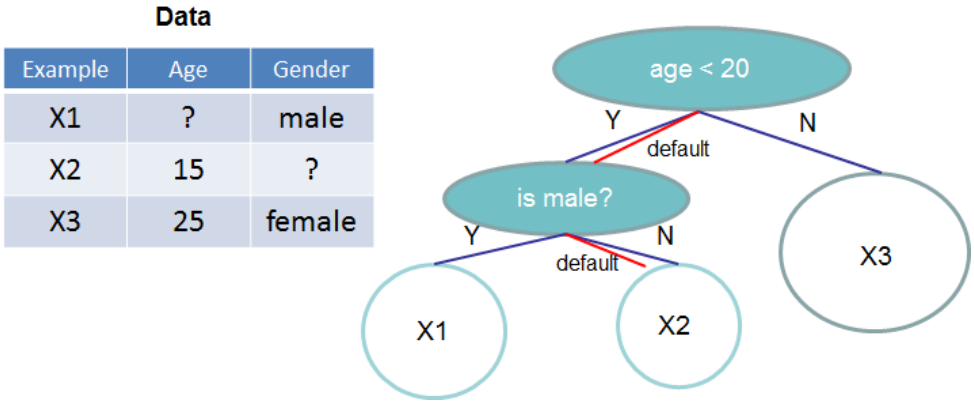
Aqui está um fator de aproximação. Intuitivamente, isso significa que há aproximadamente $1/\varepsilon$ pontos candidatos. Aqui cada ponto de dados é ponderado por h_i. Para ver porque h_i representa o peso, pode-se reescrever a Equação 2.5 como:

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(x_i) - g_i/h_i)^2 + \Omega(f_t) + constante, \tag{2.13}$$

que é exatamente a perda ao quadrado ponderada por g_i/h_i e peso h_i. Para grandes conjuntos de dados não é trivial encontrar divisões candidatas que satisfaçam os critérios. A fim de solucionar este problema, introduz-se um novo algoritmo de esboço de quantil ponderado distribuído que pode lidar com dados ponderados com uma garantia teórica comprovável. A ideia geral é propor uma estrutura de dados que suporte operações de mesclagem e poda, com cada operação comprovada para manter um certo nível de precisão (CHEN; GUESTRIN, 2016).

Em muitos problemas do mundo real, é bastante comum que a entrada x seja esparsa. Existem diversas causas possíveis para a dispersão e é importante tornar o algoritmo ciente do padrão de esparsidade nos dados. Para isso, Chen e Guestrin (2016) proporam adicionar uma direção padrão em cada nó da árvore x, que é mostrada na Figura 3. Neste caso, *'age'* significa a 'idade', *'gender'* o 'gênero', *'male'* 'masculino', *'female'* traduz-se para 'feminino' e *'default'* pode ser compreendido como 'padrão'.

Figura 3 – Estrutura em árvore com direções padrão



Fonte: (CHEN; GUESTRIN, 2016)

Quando falta um valor na matriz esparsa x, a instância é classificada na direção padrão. Existem duas escolhas de direção padrão em cada galho. As direções padrões ideais são aprendidas a partir dos dados. O algoritmo é mostrado na Figura 4. Como o saída o algoritmo obtém o *split* (em português, divisão) e direções padrão com ganho máximo.

Figura 4 – Descoberta de divisão com reconhecimento de esparsidade

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

Fonte: (CHEN; GUESTRIN, 2016)

A principal melhoria é visitar apenas as entradas não faltantes I_k. O algoritmo apresentado trata a não presença como um valor ausente e aprende a melhor direção para lidar com valores ausentes. O mesmo algoritmo também pode ser aplicado quando a ausência corresponde a um valor especificado pelo usuário, limitando a enumeração apenas a soluções consistentes.

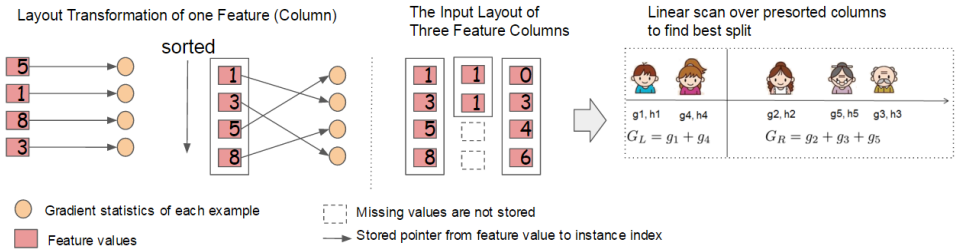
O XGBoost lida com todos os padrões de esparsidade de maneira unificada. O método proposto por Chen e Guestrin (2016) explora a escassez para tornar a complexidade computacional linear ao número de entradas não faltantes na entrada.

A parte mais demorada do aprendizado em árvore é colocar os dados de maneira ordenada. Pra reduzir o custo de ordenação, Chen e Guestrin (2016) proporam armazenar os dados em unidades *in-memory*, intituladas *block* (bloco). Os dados de cada bloco são armazenados no formato de coluna compactada (CSC), com cada coluna classificada pelo valor do atributo correspondente. Esse *layout* (ou em português, leiaute) de dados de entrada só precisa ser calculado uma vez antes do treinamento e pode ser reutilizado e iterações posteriores.

No algoritmo guloso exato, Chen e Guestrin (2016) armazenaram todo o conjunto de dados em um único bloco e executaram o algoritmo de pesquisa dividida por varredura inicial nas entradas pré-ordenadas. Fizeram a descoberta de divisão de todas as folhas coletivamente, de modo que uam varredura no bloco coletará as estatísticas dos candidatos divididos em todos os ramos da folha.

A Figura 5 demonstra como o conjunto de dados foi transformado e encontrada a divisão ideal utilizando a estrutura de blocos. À esquerda está a transformação de *layout* de um atributo (coluna), onde os círculos laranjas são as estatísticas de gradiente de cada exemplo e os retângulos vermelhos são valores dos atributos. Ao centro está o *layout* de entrada de três colunas de atributos. Por fim, à direita está a varredura linear em colunas pré-ordenadas para encontrar a melhor divisão.

Figura 5 – Estrutura de blocos para aprendizado paralelo



Fonte: (CHEN; GUESTRIN, 2016)

A estrutura de blocos também ajuda ao usar os algoritmos de aproximação. Diversos blocos podem ser usados messe caso, com cada bloco correspondendo a um subconjunto de linhas no conjunto de dados. Blocos diferentes podem ser distribuídos entre máquinas ou armazenados em disco na configuração *out-of-core* (fora do núcleo). Usando a estrutura classificada, a etapa de descoberta de quantil torna-se uma varredura linear sobre as colunas classificadas. Isso é especialmente valioso para algoritmos de propostas locais, onde os candidatos são gerados com frequência em cada filial. A pesquisa binária na agregação de histogramas também se torna um algoritmo de estilo de mesclagem de tempo linear (CHEN; GUESTRIN, 2016).

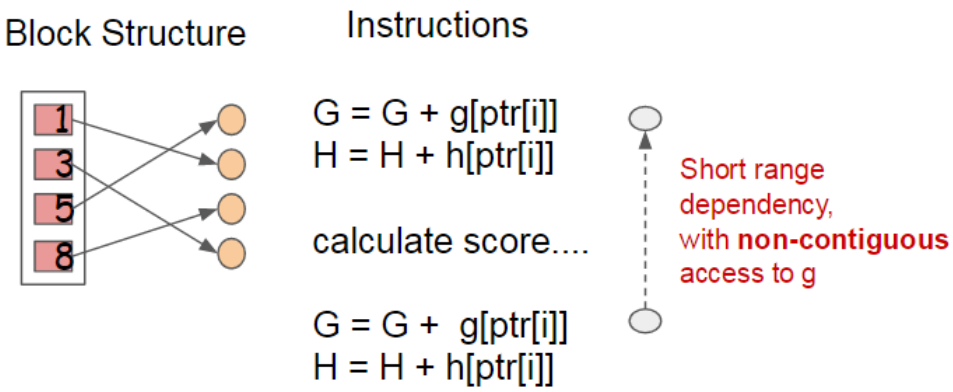
A coleta de estatísticas para cada coluna pode ser paralelizada, fornecendo um algoritmo paralelo para encontrar divisão. É importante ressaltar que a estrutura do bloco de colunas também suporta subamostragem de colunas, pois é fácil selecionar um subconjunto de colunas em um bloco.

Seja d a profundidade máxima da árvore e K o número total de árvores, para o algoritmo guloso exato, a complexidade de tempo do espaço original algoritmo ciente é $O(Kd||x||_0 \log n)$. Aqui usa-se $||x||_0$ para denotar o número de entrada não faltantes nos dados de treinamento. Por outro lado, o aumento de árvore na estrutura de blocos custa apenas $O(Kd||x||_0+||x||_0 \log n)$. $O(||x||_0$
$\log n)$ é aquele custo de pré-processamento de tempo que pode ser amortizado. Esta análise mostra que a estrutura de blocos ajuda a economizar um fator $\log n$, que é significativo quando

n é grande. Para o algoritmo aproximado, a complexidade de tempo do algoritmo original com busca binária é $O(Kd||x||_0 \log q)$. q é o número de candidatos a propostas no conjunto de dados. Enquanto q é geralmente entre 32 e 100, o fator log ainda introduz a sobrecarga. Usando a estrutura de blocos, pode-se reduzir o tempo para $O(Kd||x||_0 + ||x||_0 \log B)$, onde B é o numero máximo de linhas em cada bloco. Novamente, pode-se salvar o adicional fator $\log q$ na computação (CHEN; GUESTRIN, 2016).

Embora a estrutura de blocos ajude a otimizar a complexidade computacional da descoberta de divisão, o novo algoritmo requer buscas indiretas de estatísticas de gradiente por índice de linha, uma vez que esses valores são acessados em ordem de atributo. Isto é, um acesso não contínuo à memória. Uma implementação *naive* (em português, ingênua) de enumeração dividida introduz uma dependência imediata de leitura/gravação entre a acumulação e operação não contínua de busca de memória, conforme pode ser observado na Figura 6. Isso desacelera a divisão (*splitting*) quando as estatísticas de gradiente não cabem no cache da CPU e ocorrem falhas de cache (CHEN; GUESTRIN, 2016).

Figura 6 – Padrão de dependência de dados de curto alcance



Fonte: (CHEN; GUESTRIN, 2016)

Na imagem acima, à esquerda está a estrutura de bloco. Ao centro, estão as instruções e cálculo de pontuação. A dependência é de curto alcance, com acesso não contíguo a g.

Para o algoritmo guloso exato, pode-se aliviar o problema por um algoritmo de pré-busca com reconhecimento de cache. Especificamente, aloca-se um *buffer* interno em cada *thread*, buscando as estatísticas de gradiente nele e, em seguida, realiza-se a acumulação em uma forma de *mini-batch* (em português, mini-lote). Esta pré-busca (*prefetching*) altera a dependência de leitura/escrita para uma dependência mais longa e ajuda a reduzir a sobrecarga de tempo de execução quando o número de linhas é grande (CHEN; GUESTRIN, 2016).

A implementação com reconhecimento de cache de algoritmo ganancioso exato é executado duas vezes mais rápido que o *naive* quando o conjunto de dados é grande. Para algoritmos aproximados, o problema é resolvido escolhendo um tamanho de bloco correto. Define-se o tamanho do bloco como o número máximo de exemplo contidos em um bloco, pois isso reflete o custo de armazenamento em cache das estatísticas de gradiente. Escolher um tamanho de bloco muito pequeno resulta uma pequena carga de trabalho para cada *thread* e leva a uma

paralelização ineficiente. Por outro lado, blocos muito grandes resultam em falhas de cache, pois as estatísticas de gradiente não cabem no cache da CPU. Uma boa escolha de tamanho de bloco equilibra esses dois fatores (CHEN; GUESTRIN, 2016).

Um dos objetivos do sistema proposto por Chen e Guestrin (2016) é utilizar totalmente os recursos de uma máquina para obter aprendizado escalável. Além de processadores e memória, é importante utilizar o espaço em disco para lidar com dados que não cabem na memória principal. Para habilitar computação *out-of-core*, os dados foram divididos em vários blocos e armazenados no disco. Durante a computação, é importante usar uma *thread* independente para pré-carregar o bloco em um *buffer* de memória principal, de modo que a computação possa ocorrer simultaneamente com a leitura do disco. No entanto, isso não resolve completamente o problema, já que a leitura do disco consome a maior parte do tempo de computação. É importante reduzir a sobrecarga e aumentar o rendimento da escrita/leitura de disco (CHEN; GUESTRIN, 2016).

Duas técnicas foram utilizadas para melhorar a computação *out-of-core*. A primeira técnica é a *block compression* (em português, compressão de bloco). O bloco é comprimido por colunas e descomprimido na memória principal por uma *thread* independente quando carregado. Isso ajuda a trocar parte da computação na descompressão pelo custo de leitura do disco. Um algoritmo de compressão de propósito geral é usado para comprimir os valores das características. Para o índice de linha, subtrai-se o índice da linha pelo índice inicial do bloco e usamos um inteiro de 16 bits para armazenar cada deslocamento. Isso requer 216 exemplos por bloco, o que foi confirmado como uma boa configuração (CHEN; GUESTRIN, 2016).

A segunda técnica consiste na desfragmentação dos dados em vários discos de forma alternativa. Um *thread* de pré-carregamento é atribuído a cada disco e busca os dados em um *buffer* na memória. A *thread* de treinamento, então, lê alternadamente os dados de cada *buffer*. Isso ajuda a aumentar o rendimento da leitura de disco quando vários discos estão disponíveis (CHEN; GUESTRIN, 2016).

O XGBoost é então um algoritmo de aprendizado de máquina que utiliza árvores de decisão como base para construir modelos. Faz uso de uma série de técnicas para melhorar a precisão do modelo, como a regularização de segundo grau, a otimização da função de perda e a estratégia de poda. Essas técnicas tornam o XGBoost um algoritmo poderoso e eficaz para a construção de modelos de aprendizado de máquina.

2.2.2 Random Forest

Criado por Breiman (2001), baseado no artigo sobre reconhecimento de caracteres escritos (AMIT; GEMAN, 1997 apud BREIMAN, 2001), o algoritmo de florestas aleatórias (em inglês, *Random Forest*) é uma combinação de preditores de árvores de modo que cada árvore depende dos valores de um vetor aleatório amostrado independente e com a mesma distribuição para todas as árvores da floresta. O erro de generalização para florestas converge para um limite à medida que o número de árvores na floresta se torna grande. O erro de generalização de uma floresta de classificadores de árvores depende da força das árvores individuais

na floresta e da correlação entre elas. As estimativas internas monitoram o erro, a força e a correlação e são usadas para mostrar a resposta ao aumento do número de atributos usados na divisão e também são usadas para medir a importância da variável. Tais ideias também são aplicáveis à regressão (BREIMAN, 2001).

Um vetor aleatório Θ_k é gerado, independente dos vetores aleatórios passados $1, ..., \Theta_{k-1}$ mas com a mesma distribuição. Uma árvore é elaborada usando o conjunto de treinamento e Θ_k, resultando em um classificador $h(x, \Theta_k)$ onde x é a entrada do vetor. Depois que um grande número de árvores é gerado, é votado na classe mais popular. Esses procedimentos são chamados de florestas aleatórias (BREIMAN, 2001).

Random Forest é um classificador que consiste em uma coleção de árvores de decisão $\{h(x, \Theta_k)\ k = 1, ...\}$ onde $\{\Theta_k\}$ são vetores aleatórios independentes igualmente distribuídos e cada árvore realiza um voto unitário para a classe mais popular na entrada x (BREIMAN, 2001).

Dado um conjunto de classificadores $h_1(x), h_2(x), ..., h_k(x)$, e com o conjunto de treinamento extraído aleatoriamente da distribuição do vetor aleatório Y, X a função de margem é definida conforme Equação 2.14 abaixo:

$$mg(X, Y) = av_k I(h_k(x) = Y) - max_{j \neq Y} av_k I(h_k(x) = j) \tag{2.14}$$

onde $I(\cdot)$ é a função indicativa. A margem mede até que ponto o número médio de votos em X, Y para a classe certa excede a média de votos para qualquer outra classe. Quanto maior a margem, maior a confiança na classificação. O erro de generalização é dado pela Equação 2.15.

$$PE^* = P_{X,Y}(mg(X, Y) < 0) \tag{2.15}$$

onde os subscritos X, Y indicam que a probabilidade é além do espaço X, Y. Em florestas aleatórias, $h_k(X) = h_k(X, \Theta_k)$. Para uma grande quantidade de árvores, segue da Lei Forte dos Grandes Números e a estrutura da árvore. A medida que o número de árvores aumenta, para quase certamente todas as sequências $\Theta_1, ...PE^*$ converge para

$$P_{X,Y}(P_\Theta(h(X, \Theta) = Y) - max_{j \neq Y} P\Theta(h(X, \Theta) = j) < 0) \tag{2.16}$$

O resultado apresentado pela Equação 2.16 explica o porquê das florestas aleatórias não superajustarem à medida que mais árvores são adicionadas, mas produzem um valor limite de erro de generalização (BREIMAN, 2001).

Para *Random Forest*, um limite superior pode ser derivado para o erro de generalização em termos de dois parâmetros que são medidas de quão precisos são os classificadores individuais e da dependência entre eles. A interação entre esses dois parâmetros dá a base o entendimento para o funcionamento das florestas aleatórias (BREIMAN, 2001).

A função de margem para uma floresta aleatória é fornecida pela equação Equação 2.17.

$$mr(X,Y) = P_{\Theta}(h(X,\Theta) = Y) - max_{j \neq Y} P_{\Theta}(h(X,\Theta) = j) \tag{2.17}$$

e a força do conjunto de classificadores $\{h(X,\Theta)\}$ é dada pela equação Equação 2.18.

$$s = E_{X,Y} mr(X,Y) \tag{2.18}$$

Assumindo $s \geq 0$, a desigualdade de Chebychev fornece a Equação 2.19.

$$PE^* \leq var(mr)/s^2 \tag{2.19}$$

Uma expressão mais reveladora para a variância mr é derivada da equação Equação 2.20.

$$\hat{j}(X,Y) = argmax_{j \neq Y} P_{\Theta}(h(X,\Theta) = j) \tag{2.20}$$

então

$$mr(X,Y) = P_{\Theta}(h(X,\Theta) = Y) - P_{\Theta}(h(X,\Theta) = \hat{j}(X,Y)) \tag{2.21}$$

$$= E_{\Theta}[I(h(X,\Theta) = Y) - I(h(X,\Theta) = \hat{j}(X,Y))] \tag{2.22}$$

A função de margem bruta é dada pela equação Equação 2.23.

$$rmg(\Theta,X,Y) = I(h(X,\Theta) = Y) - I(h(X,\Theta) = \hat{j}(X,Y)) \tag{2.23}$$

Assim, $mr(X,Y)$ é a expectativa de $rmg(\Theta,X,Y)$ em relação à Θ. Para qualquer função f a identidade

$$[E_\Theta f((\Theta))]^2 = E_{\Theta,\Theta'} f(\Theta) f(\Theta') \tag{2.24}$$

detém onde Θ, Θ' são independentes com a mesma distribuição, o que implica que

$$mr(X,Y)^2 = E_{\Theta,\Theta'} rmg(\Theta,X,Y) rmg(\Theta',X,Y) \tag{2.25}$$

Utilizando a equação Equação 2.25

$$var(mr) = E_{\Theta,\Theta'}(cov_{X,Y} rmg(\Theta,X,Y) rmg(\Theta',X,Y)) \tag{2.26}$$

$$= E_{\Theta,\Theta'}(\rho(\Theta,\Theta') sd(\Theta')) \tag{2.27}$$

onde $\rho(\Theta,\Theta'$ é a correlação entre $rmg(\Theta,X,Y)$ e $rmg(\Theta',X,Y)$ contendo Θ,Θ' fixados e $sd(\Theta)$ é o desvio padrão de $rmg(\Theta,X,Y)$ contendo Θ fixado. Então,

$$var(mr) = \overline{\rho}(E_\Theta sd(\Theta))^2 \tag{2.28}$$

$$\leq \overline{\rho} E_\Theta var(\Theta)) \tag{2.29}$$

onde $\overline{\rho}$ é o valor médio da correlação, isso é,

$$\overline{\rho} = E_{\Theta,\Theta'}(\rho(\Theta,\Theta')) sd(\Theta) sd(\Theta') / E_{\Theta,\Theta'}(sd(\Theta) sd(\Theta')) \tag{2.30}$$

Descrevendo

$$E_\Theta var(\Theta) \leq E_\Theta (E_{X,Y} rmg(\Theta,X,Y))^2 - s^2 \tag{2.31}$$

$$\leq 1 - s^2 \tag{2.32}$$

Reunindo as equações Equação 2.19 e Equação 2.28 e Equação 2.31 resulta no limite superior para o erro de generalização, que é dado por:

$$PE^* \leq \overline{\rho}(1 - s^2)/s^2 \tag{2.33}$$

Breiman (2001) afirma que embora o limite provavelmente seja livre, este cumpre a mesma função sugestiva para florestas aleatórias que os limites do tipo VC fazem para outros tipos de classificadores. Mostra que os dois *ingredients* (em tradução livre, ingredientes) envolvidos no erro de generalização para florestas aleatórias são a força dos classificadores individuais na floresta e a correlação entre eles em termos das funções de margem bruta. A razão $c/s2$ é a correlação dividida pelo quadrado da força. Ao entender o funcionamento das florestas aleatórias, essa proporção será útil - quanto menor, melhor (BREIMAN, 2001).

A razão $c/s2$ para uma floresta aleatória é definida como

$$c/s2 = \overline{\rho}/s^2 \tag{2.34}$$

Existem simplificações na situação de duas classes. A função margem é

$$mr(X, Y) = 2P_{\Theta}(h(X, \Theta) = Y) - 1 \tag{2.35}$$

A exigência de que a força seja positiva (Equação 2.19) torna-se semelhante à condição familiar de aprendizagem fraca $E_{X,Y}P_{\Theta}(h(X, \Theta) = Y) > .5$. A função de margem bruta é $2I(h(X, \Theta) = Y) - 1$ e a correlação $\overline{\rho}$ está entre $I(h(X, \Theta) = Y)$ e $I(h(X, \Theta') = Y)$. Em particular, se os valores de Y forem tomados para ser $+1$ e -1, extensão

$$\overline{\rho} = E_{\Theta,\Theta'}[\overline{\rho}(h(., \Theta), h(., \Theta'))] \tag{2.36}$$

de modo que $\overline{\rho}$ é a correlação entre dois membros diferentes da floresta em média ao longo da distribuição Θ, Θ' (BREIMAN, 2001).

Para mais de duas classes, a medida de força definida pela equação Equação 2.18 depende da floresta, bem como das árvores individuais, pois é a floresta que determina $\tilde{j}(X, Y)$ (BREIMAN, 2001).

Outra abordagem é possível:

$$PE^* = P_{X,Y}(P_{\Theta}(h(X, \Theta) = Y) - max_{j \neq Y} P_{\Theta}(h(X, \Theta) = j)) < 0 \tag{2.37}$$

$$\leq \sum_j P_{X,Y}(P_{\Theta}(h(X, \Theta) = Y) - P_{\Theta}(h(X, \Theta) = j)) < 0 \tag{2.38}$$

Definindo

$$s_j = E_{X,Y}(P_\Theta(h(X,\Theta) = Y) - P_\Theta(h(X,\Theta) = j)) \tag{2.39}$$

para ser a força do conjunto de classificadores $h(x,\Theta)$ em relação à classe j. A definição de força não depende da floresta. Utilizando a desigualdade de Chebychev, assumindo que todo $s_j > 0$ leva a

$$PE^* \leq \sum_j var(P_\Theta(h(X,\Theta) = Y) - P_\Theta(h(X,\Theta) = j))s_j^2 \tag{2.40}$$

e usando identidades semelhantes às usadas para derivas a Equação 2.28, as variâncias na Equação 2.40 podem ser expressas em termos de correlações médias.

Random Forest é uma ferramenta eficaz na precisão. Por causa da lei dos grandes números eles não superajustam. Injetar o tipo certo de aleatoriedade os torna classificadores precisos e regressores. Além disso, a estrutura em termos de força dos preditores individuais e suas correlações fornece informações sobre a capacidade da floresta aleatória realizar previsões. O uso da estimativa *out-of-bag* (em português, fora da bolsa) torna concretos os valores teóricos da resistência e correlação (BREIMAN, 2001).

As florestas dão resultados competitivos com *boosting* e *bagging* adaptativo, mas não alteram progressivamente o conjunto de treinamento. Sua precisão indica que elas agem para reduzir o viés. Entradas e características aleatórias produzem bons resultados na classificação, porém menos na regressão. Os únicos tipos de aleatoriedade utilizados no estudo realizado por Breiman (2001) são *bagging* e características aleatórias.

2.2.3 Logistic Regression

Apesar do seu nome, a regressão logística é implementada como um modelo linear para classificação ao invés de regressão em termos da nomenclatura *scikit-learn/ML*. A logística a regressão também é conhecida na literatura como regressão *logit*, *maximum-entropy classification* (em português, classificação de entropia máxima) (*MaxEnt*) ou como classificador *log-linear*. Neste modelo, as probabilidades que descrevem os possíveis resultados de um único ensaio são modelados usando uma função logística (PEDREGOSA et al., 2011).

Essa implementação pode se adequar à logística binária, de um *one-vs-rest* ou regressão multinomial com opcional l_1, l_2, ou regularização *Elastic-Net*. A regressão logística é um caso especial de *Generalized Linear Models* (ou em português, Modelos Lineares Generalizados) com uma distribuição condicional Binomial/Bernoulli e um *link Logit*. A saída numérica da logística a regressão, que é a probabilidade prevista, pode ser usada como classificador

aplicando-lhe um limite (por padrão 0,5). Assim é implementado em *scikit-learn*, por isso espera um alvo categórico, tornando a Regressão Logística um classificador (PEDREGOSA et al., 2011).

Para facilidade notacional, assumimos que o destino y_i assume valores no conjunto $\{0, 1\}$ para o ponto de dados i. Uma vez instalado, o método *predict_proba* de *LogisticRegression* prevê a probabilidade da classe positiva $P(y_i = 1|X_i)$ como

$$\hat{p}(X_i) = expit(X_i w + w_0) = \frac{1}{1 + exp(-X_i w - w_0)} \tag{2.41}$$

Como um problema de otimização, regressão logística de classe binária com termos de regularização $r(w)$ minimiza a seguinte função de custo:

$$\min_w C \sum_{i=1}^{n} (-y_i \log(\hat{p}(X_i)) - (1 - y_i) log(1 - \hat{p}(X_i))) + r(w) \tag{2.42}$$

Há quatro opções para o prazo de regularização do termo $r(w)$ através do parâmetro *'penalty'* (em português, penalidade), conforme demonstrado na Figura 7.

Figura 7 – Penalidades para o caso binário

penalty	$r(w)$
`None`	0
ℓ_1	$\|w\|_1$
ℓ_2	$\frac{1}{2}\|w\|_2^2 = \frac{1}{2} w^T w$
`ElasticNet`	$\frac{1-\rho}{2} w^T w + \rho \|w\|_1$

Fonte: (PEDREGOSA et al., 2011)

Para *ElasticNet*, ρ(que corresponde ao *'l1_ratio'* parâmetro) controla a força da regularização l_1 *versus* regularização l_2. *Elastic-Net* é equivalente a l_1 quando $\rho = 1$ e equivalente a l_2 quando $\rho = 0$.

O caso binário pode ser estendido para K classes que conduzem à regressão logística multinomial. É possível parametrizar um modelo de classificação K usando apenas $K - 1$ vetores de peso, deixando uma probabilidade de classe totalmente determinado pelas probabilidades da outra classe, aproveitando o fato de que todos as probabilidades de classe devem somar um. Escolha-se super-parametrizar o modelo usando K vetores de peso para facilitar a implementação e preservar o viés indutivo simétrico em relação à ordenação das classes. Este efeito se torna especialmente importante ao utilizar a regularização. A escolha da super-parametrização pode ser prejudicial para modelos não penalizados, assim, a solução pode não ser única (PEDREGOSA et al., 2011).

Seja $y_i \in 1, ..., K$ a variável de destino codificada no rótulo (ordinal) para observação i. Em vez de um único vetor de coeficiente, agora tem-se uma matriz de coeficientes W onde cada vetor linha W_k corresponde à classe k. O objetivo é prever as probabilidades de classe $P(y_i = k|X_i)$ através de *'predict_proba'* como:

$$\hat{p}_k(X_i) = \frac{exp(X_i W_k + W_{0,k})}{\sum_{l=1}^{K-1} exp(X_i W_l + W_{0,l})} \tag{2.43}$$

O objetivo para a otimização torna-se

$$\min_W C \sum_{i=1}^{n} \sum_{k=0}^{K-1} [y_i = k] \log(\hat{p}_k(X_i)) + r(W) \tag{2.44}$$

Onde $[P]$ representa o *bracket Iverson* (em tradução livre, colchete de Iverson) que avalia para 0 se P é falso, caso contrário, é avaliado como 1. Há quatro opções para o termo regularização $r(W)$ através do argumento *'penalty'*, conforme pode ser visto na Figura 8.

Figura 8 – Penalidades para o caso multinomial

penalty	$r(W)$
`None`	0
ℓ_1	$\|W\|_{1,1} = \sum_{i=1}^{n} \sum_{j=1}^{K} \|W_{i,j}\|$
ℓ_2	$\frac{1}{2}\|W\|_F^2 = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{K} W_{i,j}^2$
`ElasticNet`	$\frac{1-\rho}{2}\|W\|_F^2 + \rho\|W\|_{1,1}$

Fonte: (PEDREGOSA et al., 2011)

Os solucionadores implementados na classe LogisticRegression são *'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag'* e *'saga'* (PEDREGOSA et al., 2011).

O solucionador *'liblinear'* usa um algoritmo de *coordinate descent* ou em português, descida de coordenadas (CD) e depende da biblioteca LIBLINEAR C++, fornecida com *scikit-learn* (PEDREGOSA et al., 2011). No entanto, o algoritmo de CD implementado em *liblinear* não pode aprender um modelo multinomial verdadeiro (multiclasse); em vez disso, o problema de otimização é decomposto de uma forma *'one-vs-rest'* de modo que classificadores binários separados sejam treinados para todas as classes. Isso acontece *under the hood* (em tradução livre, nos bastidores), de modo que as instâncias '*LogisticRegression*' que usam esse solucionador se comportam como classificadores multiclasse. Para regularização l_1 *'sklearn.svm.l1_min_c'* permite calcular o limite inferior para C, a fim de obter um modelo não "nulo"(todos os atributos pesos a zero).

Os solucionadores *'lbfgs'*, *'newton-cg'* e *'sag'* suportam apenas regularização l_2 ou sem regularização e convergem mais rapidamente para alguns dados de alta dimensão. Definir

'multi_class' como *"multinomial'* com esses solucionadores aprende um verdadeiro modelo de regressão logística multinomial, o que significa que suas estimativas de probabilidade devem ser melhor calibradas do que a configuração padrão *'one-vs-rest'* (PEDREGOSA et al., 2011).

O solucionador *'sag'* usa a descida *Stochastic Average Gradient*. É mais rápido do que outros solucionadores para grandes conjuntos de dados, quando o número de amostras e o número de atributos são grandes.

O solucionador *'saga'* é uma variante do *'sag'* que também suporta o não suave 'penalty = "l1"'. Este é, portanto, o solucionador de escolha para a regressão logística multinomial esparsa. É também o único solucionador que suporta 'penalty = "elasticnet"' (PEDREGOSA et al., 2011).

O *'lbfgs'* é um algoritmo de otimização que se aproxima do algoritmo *Broyden–Fletcher–Goldfarb–Shanno*, que pertence aos métodos *quasi-Newton*. Como tal, ele pode lidar com uma ampla gama de dados de treinamento diferentes e, portanto, é o solucionador padrão. Seu desempenho, no entanto, sofre em conjuntos de dados mal dimensionados e em conjuntos de dados com atributos categóricos codificados *one-hot* com categorias raras (PEDREGOSA et al., 2011).

O solucionador *'newton-cholesky'* é um solucionador exato de Newton que calcula a matriz hessiana e resolve o sistema linear resultante. É uma escolha muito boa para $'n\_samples' >>' n\_features'$, mas tem algumas deficiências: Apenas a regularização l_2 é suportada. Além disso, como a matriz hessiana é computada explicitamente, o uso da memória tem uma dependência quadrática *'n_featurestanto'* em quanto em *'n_classes'*. Como consequência, apenas o esquema *one-vs-rest* é implementado para o caso multiclasse (PEDREGOSA et al., 2011).

A Figura 9 resume as penalidades suportadas por cada solucionador:

Figura 9 – Solucionadores

Penalidades	**'lbfgs'**	**'libli-near'**	**'new-ton-cg'**	**'newton-cholesky'**	**'sag'**	**'saga'**
Multinomial + penali-dade L2	sim	não	sim	não	sim	sim
OVR + penalidade L2	sim	sim	sim	sim	sim	sim
Multinomial + penali-dade L1	não	não	não	não	não	sim
OVR + penalidade L1	não	sim	não	não	não	sim
Elastic-Net	não	não	não	não	não	sim
Nenhuma penalidade	sim	não	sim	sim	sim	sim
Comportamentos						
Penalizar a interceptação (ruim)	não	sim	não	não	não	não
Mais rápido para gran-des conjuntos de dados	não	não	não	não	sim	sim
Robusto para conjuntos de dados não dimensionados	sim	sim	sim	sim	não	não

Fonte: Adaptado de (PEDREGOSA et al., 2011)

O solucionador *'lbfgs'* é usado por padrão por sua robustez. Para grandes conjuntos de dados, o solucionador *'saga'* geralmente é mais rápido. Para conjuntos de dados grandes, pode-se considerar o uso *'SGDClassifier'* com *'loss="log_loss"'*, que pode ser ainda mais rápido, mas requer mais ajustes (PEDREGOSA et al., 2011).

Pode haver uma diferença nas pontuações obtidas entre *'LogisticRegression'* com *'solver = liblinear'* ou *'LinearSVC'* e a biblioteca *liblinear* externa diretamente, quando *'fit_intercept = False'* e o ajuste *'coef_'* (ou) os dados a serem previstos são zeros. Isso ocorre porque para a(s) amostra(s) com *'decision_function'* zero, *LogisticRegression* e *'LinearSVC'* prediz a classe negativa, enquanto *liblinear* prediz a classe positiva. Observa-se que um modelo com *'fit_intercept = False'* e com muitas amostras com *'decision_function'* zero provavelmente será um modelo inadequado e ruim e é recomendável definir *'fit_intercept = True'* e aumentar o *'intercept_scaling'* (PEDREGOSA et al., 2011).

'LogisticRegressionCV' implementa Regressão Logística com suporte integrado de validação cruzada, para encontrar os parâmetros 'C' e 'l1_ratio' ideais de acordo com o atributo *'scoring'*. Os solucionadores *'newton-cg'*, *'sag'*, *'saga'* e *'lbfgs'* são mais rápidos para dados densos de alta dimensão, devido a *warm-starting* (em tradução livre, inicialização quente) (PEDREGOSA et al., 2011).

2.3 Métricas

Ao construir um classificador utilizando aprendizado de máquina, a fim de assegurar o quão bom é o modelo de predição, algumas métricas podem ser utilizadas para avaliação de acordo com o problema analisado (SOUZA, 2019).

2.3.1 Acurácia

A acurácia (em inglês, *accuracy*) possui o objetivo de descobrir o quão frequente o classificador está correto. É a proporção de predições corretas sem considerar o que é positivo ou negativo, mas sim o acerto total (ANDRADE, 2019) e é dada pela Equação 2.45. O total de predições pode ser compreendido como a soma de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos (MENESES, 2021; POWERS, 2020).

$$\text{Acurácia} = \frac{\text{Verdadeiros positivos} + \text{Verdadeiros negativos}}{\text{Total de Predições}} \tag{2.45}$$

2.3.2 Precisão

A precisão (em inglês, *precision*) ou confiança (em inglês, *confidence*) denota a proporção de casos classificados como positivos que são corretamente verdadeiros positivos (POWERS, 2020).

Traduzindo em fórmula temos a Equação 2.46:

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}} \tag{2.46}$$

2.3.3 Sensibilidade

A revocação (ou em inglês *recall*) é a proporção de verdadeiros positivos que o classificador acertou. Pode também ser compreendido como sensibilidade do modelo, a proporção de verdadeiros positivos, ou seja, avalia a capacidade do modelo classificar um indivíduo como evento (predição = 1) dado que realmente ele é evento de interesse (variável resposta = 1) (ANDRADE, 2019; POWERS, 2020).

Em fórmula temos a Equação 2.47 abaixo:

$$\text{Sensibilidade} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos negativos}} \tag{2.47}$$

2.3.4 Especificidade

A especificidade é a proporção de verdadeiros negativos, ou seja, avalia a capacidade do modelo predizer um indivíduo como não evento dado que ele realmente é não evento (ANDRADE, 2019). Em fórmula temos a Equação 2.48:

$$\text{Especificidade} = \frac{\text{Verdadeiros negativos}}{\text{Verdadeiros negativos} + \text{Falsos positivos}} \tag{2.48}$$

2.3.5 F1 Score

Também conhecida como *F-score*, essa métrica surge a partir da combinação entre precisão e sensibilidade de modo a trazer um único número que indique a qualidade geral do modelo. É a média harmônica entre precisão e sensibilidade (MENESES, 2021). A Equação 2.49 apresenta esta combinação:

$$\text{F1 Score} = \frac{2 \times \text{Precisão} \times \text{Sensibilidade}}{\text{Precisão} + \text{Sensibilidade}} \tag{2.49}$$

2.4 Validação cruzada: K-fold

Na técnica de validação cruzada K-fold (BURMAN, 1989), a amostra d é dividida em K subamostras ($d_1, d_2, ..., d_k$), de tamanho similar m_k, em que $\sum_{k=1}^{K} m_k = n$. O processo é repetido K vezes, sendo que a cada iteração a amostra de validação será dada por d_k, com $k = 1, 2.., K$, e a amostra de treino para a criação do preditor será o conjunto das outras $K - 1$ partes, ou seja, $d_{(-k)} = \{d_1, d_2, ..., d_{k-1}, d_{k+1}, ..., d_K\}$. Assim, ao final dos K passos, será usado todos os dados tanto na parte de treino, quanto na parte de validação (CUNHA, 2019).

2.5 SMOTE

SMOTE significa *Synthetic Minority Over-sampling Technique*, que em português pode ser traduzido como Técnica de Sobre-amostragem Sintética da Classe Minoritária e fornece uma nova abordagem para *oversampling*. É uma técnica de balanceamento de dados usada em problemas de classificação em que a classe minoritária tem um número significativamente menor de exemplos do que a classe majoritária. O SMOTE pode melhorar a precisão dos classificadores para uma classe de menor quantidade de registros. (CHAWLA et al., 2002).

Proposta por Chawla, Bowyer, Hall e Kegelmeyer (2002) a técnica de balanceamento SMOTE trabalha criando novos exemplos sintéticos da classe minoritária, com base nos exemplos existentes dessa classe, isto é, é utilizado uma abordagem de sobreamostragem na qual a classe minoritária é sobreamostrada criando exemplos 'sintéticos' em vez de sobreamostragem com substituição. São gerados exemplos sintéticos de maneira menos específica, operando em 'espaço de atributos' em vez de 'espaço de dados'. A classe minoritária é superamostrada tomando cada amostra de classe minoritária e introduzindo exemplos sintéticos ao longo dos segmentos de linha que unem qualquer/todos os k vizinhos mais próximos da classe minoritária. Dependendo da quantidade de *oversampling* necessária, os vizinhos dos k vizinhos mais próximos são escolhidos aleatoriamente (CHAWLA et al., 2002).

As amostras sintéticas são geradas da seguinte maneira: pega-se a diferença entre o vetor de características (amostra) em consideração e seu vizinho mais próximo. Multiplica-se essa diferença por um número aleatório entre 0 e 1 e adicione-o ao vetor de atributos em consideração. Isso causa a seleção de um ponto aleatório ao longo do segmento de linha entre dois atributos específicos. Essa abordagem efetivamente força a região de decisão da classe minoritária a se tornar mais geral (CHAWLA et al., 2002).

O algoritmo apresentado na Figura 10, é o pseudocódigo para SMOTE. Como entrada o algoritmo recebe o número de amostras de classes minoritárias T, a quantidade de SMOTE N e o número de vizinhos mais próximos k (CHAWLA et al., 2002).

Figura 10 – Algoritmo do SMOTE

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples
1. (∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
8. $k =$ Number of nearest neighbors
9. $numattrs =$ Number of attributes
10. $Sample[\,][\,]$: array for original minority class samples
11. $newindex$: keeps a count of number of synthetic samples generated, initialized to 0
12. $Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the $nnarray$
15. Populate(N, i, $nnarray$)
16. **endfor**

Populate(N, i, $nnarray$) (∗ *Function to generate the synthetic samples.* ∗)
17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** $numattrs$
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21. Compute: $gap =$ random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (∗ *End of Populate.* ∗)
End of Pseudo-Code.

Fonte: (CHAWLA et al., 2002)

A Figura 11 mostra um exemplo de cálculo de amostras sintéticas aleatórias. A quantidade de sobreamostragem é um parâmetro do sistema, e uma série de curvas ROC pode ser gerada para diferentes populações e análises ROC realizadas (CHAWLA et al., 2002).

Figura 11 – Exemplo de geração de exemplos sintéticos (SMOTE)

Considere uma amostra (6,4) e seja (4,3) seu vizinho mais próximo.
(6,4) é a amostra para a qual os k vizinhos mais próximos estão sendo identificados.
(4,3) é um de seus k vizinhos mais próximos.

f1_1 = 6	f2_1 = 4	f2_1 - f1_1 = -2
f1_2 = 4	f2_2 = 3	f2_2 - f1_2 = -1

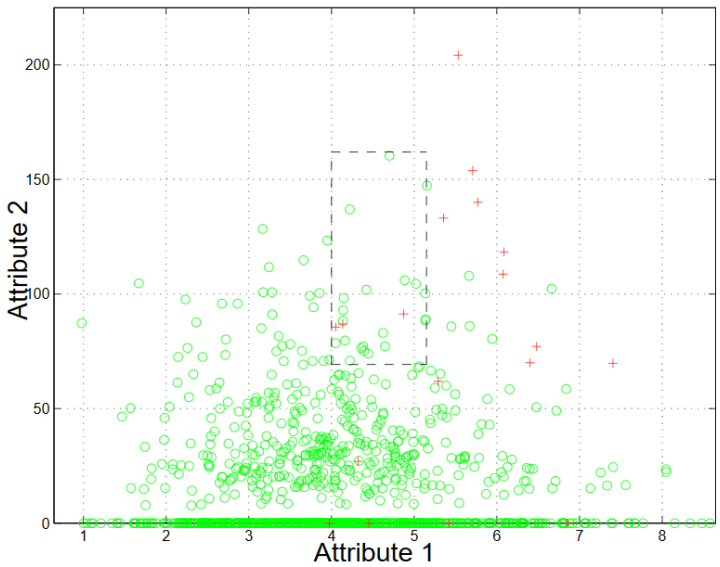
As novas amostras serão geradas como:
(f1', f2') = (6, 4) + rand(0-1) * (-2,-1)

onde 'rand(0-1)' gera um número aleatório entre 0 e 1.

Fonte: Adpatado de (CHAWLA et al., 2002)

Os exemplos sintéticos fazem com que o classificador crie regiões de decisão maiores e menos específicas, conforme mostrado pelas linhas tracejadas na Figura 12, em vez de regiões menores e mais específicas. Regiões mais gerais agora são aprendidas para as amostras de classes minoritárias, em vez daquelas que estão sendo incluídas pelas amostras de classes majoritárias ao seu redor. O efeito é que as árvores de decisão generalizam melhor (CHAWLA et al., 2002).

Figura 12 – 2 atributos, com 10% de dados do conjunto de dados original

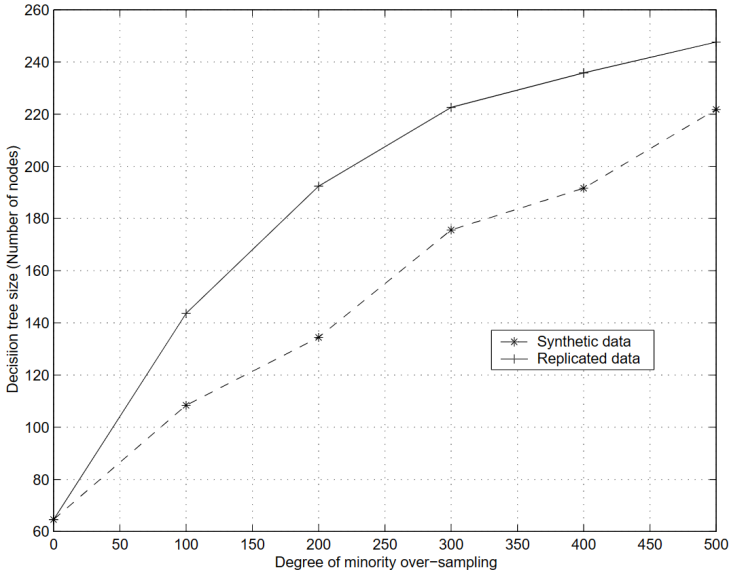


Fonte: (CHAWLA et al., 2002)

A Figura 13 e Figura 14 comparam a sobreamostragem minoritária com substituição e SMOTE. Os termos *'synthetic data'* e *'replicated data'* podem ser compreendidos como 'dados sintéticos' e 'dados replicados', respectivamente. Na Figura 13 o eixo Y representa o tamanho da árvore de decisão (número de nós) e o eixo X representa o grau de *oversampling* minoritário. Já para a Figura 14 o eixo Y e X representam a porcentagem minoritária correta e o grau

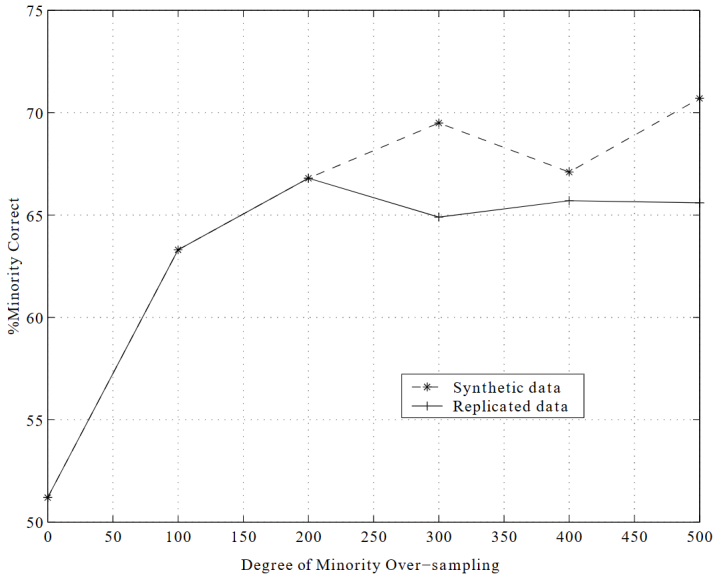
de *oversampling*minoritário, respectivamente.

Figura 13 – Comparação dos tamanhos das árvores de decisão para sobreamostragem replicada e SMOTE



Fonte: (CHAWLA et al., 2002)

Figura 14 – Comparação de % de minoria correta para sobreamostragem replicada e SMOTE



Fonte: (CHAWLA et al., 2002)

A técnica de SMOTE ajuda a evitar problemas de desequilíbrio de classe, que podem levar a modelos de aprendizado de máquina enviesados e menos precisos para a classe minoritária. O SMOTE é uma técnica eficaz para melhorar a precisão do modelo em problemas de classificação desequilibrados (FERNÁNDEZ et al., 2018).

2.6 Undersampling

Dado um conjunto de dados original S, algoritmos de geração de protótipos irão gerar um novo conjunto S' onde $|S'| < |S|$ e $S' \not\subset S$. Em outras palavras, a técnica de geração de protótipos reduzirá o número de amostras nas classes-alvo, mas as amostras restantes são geradas — e não selecionadas — do conjunto original (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Alguns algoritmos de *undersampling* serão apresentados a seguir.

O algoritmo *'ClusterCentroids'* faz uso de *K-means* para reduzir o número de amostras. Portanto, cada classe será sintetizada com os centróides do método *K-means* em vez das amostras originais, como pode ser observado na Figura 15.

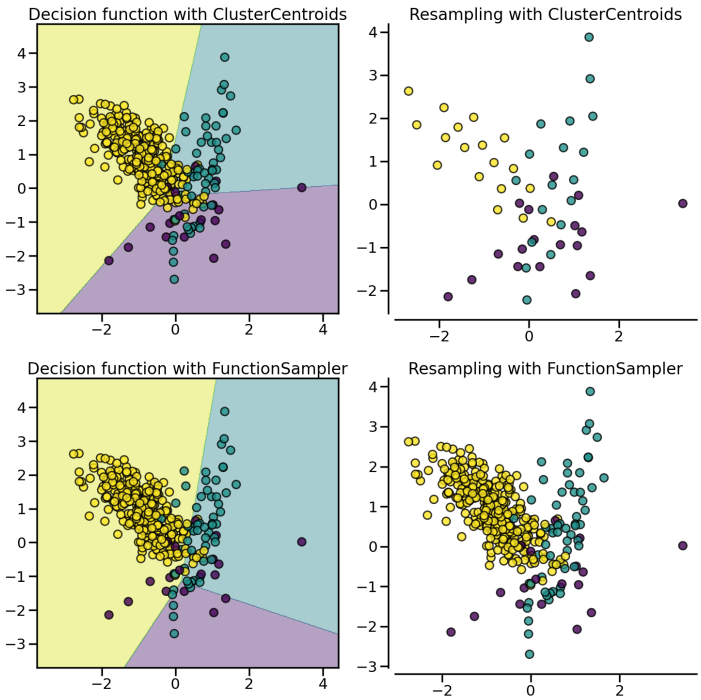
Figura 15 – Algoritmo de geração de protótipos

```
>>> from collections import Counter
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
...                            n_redundant=0, n_repeated=0, n_classes=3,
...                            n_clusters_per_class=1,
...                            weights=[0.01, 0.05, 0.94],
...                            class_sep=0.8, random_state=0)
>>> print(sorted(Counter(y).items()))
[(0, 64), (1, 262), (2, 4674)]
>>> from imblearn.under_sampling import ClusterCentroids
>>> cc = ClusterCentroids(random_state=0)
>>> X_resampled, y_resampled = cc.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 64), (2, 64)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

A Figura 16 ilustra essa subamostragem. No lado esquerdo está a função de decisão com *'ClusterCentroids'* e abaixo com *'FunctionSampler'*. À direita está o resultado após reamostragem (*resampling*). Esta observação é similar para a Figura 18, porém ao invés de *'ClusterCentroids'*, está *'RandomUnderSampler'*.

Figura 16 – Reamostragem usando *'ClusterCentroids'* e *'FunctionSampler'*



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

'ClusterCentroids' oferece uma maneira eficiente de representar o cluster de dados com um número reduzido de amostras. Lembre-se de que esse método requer que seus dados sejam agrupados em clusters. Além disso, o número de centróides deve ser definido de forma que os clusters subamostrados sejam representativos do original (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Ao contrário dos algoritmos de geração de protótipos, os algoritmos de seleção de protótipos selecionarão amostras do conjunto original S. Portanto, S' é definido como $|S'| < |S|$ e $S' \subset S$ (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Além disso, esses algoritmos podem ser divididos em dois grupos: (i) as técnicas de subamostragem controlada e (ii) as técnicas de subamostragem de limpeza. O primeiro grupo de métodos permite uma estratégia de subamostragem na qual o número de amostras em S' é especificado pelo usuário. Por outro lado, as técnicas de subamostragem de limpeza não permitem essa especificação e são destinadas à limpeza do espaço de atributos (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Já o algoritmo *'RandomUnderSampler'* é uma maneira rápida e fácil de equilibrar os dados selecionando aleatoriamente um subconjunto de dados para as classes de destino. Seu algoritmo pode ser observado na Figura 17.

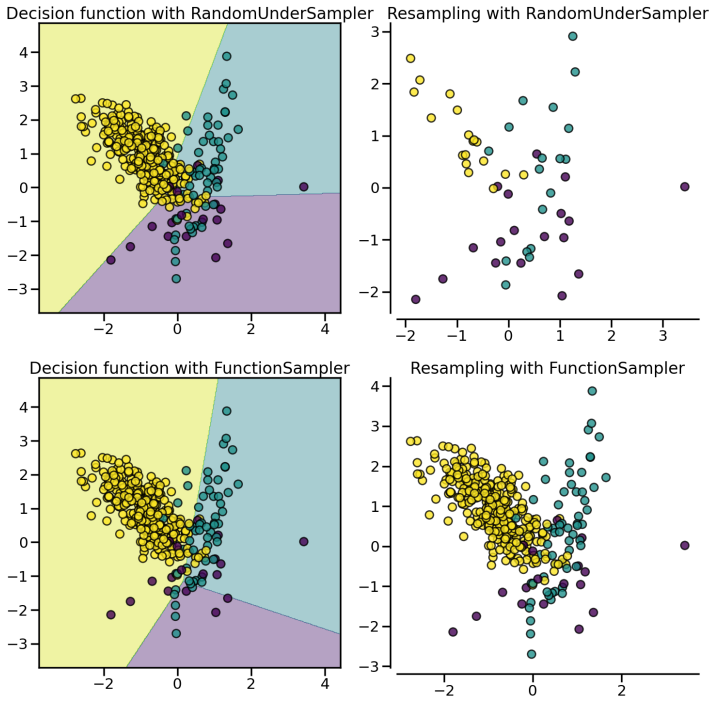
Figura 17 – Algortimo RandomUnderSampler

```
>>> from imblearn.under_sampling import RandomUnderSampler
>>> rus = RandomUnderSampler(random_state=0)
>>> X_resampled, y_resampled = rus.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 64), (2, 64)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O algoritmo acima produz a Figura 18:

Figura 18 – Reamostragem usando *'RandomUnderSampler'* e *'FunctionSampler'*



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

'RandomUnderSampler' permite inicializar os dados definindo *'replacement'* como *'True'*. A reamostragem com várias classes é realizada considerando independentemente cada classe alvo, conforme pode ser observado no algortimo presente na Figura 19.

Figura 19 – Algortimo RandomUnderSampler com várias classes

```
>>> import numpy as np
>>> print(np.vstack([tuple(row) for row in X_resampled]).shape)
(192, 2)
>>> rus = RandomUnderSampler(random_state=0, replacement=True)
>>> X_resampled, y_resampled = rus.fit_resample(X, y)
>>> print(np.vstack(np.unique([tuple(row) for row in X_resampled], axis=0)).shape)
(181, 2)
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Além disso, *'RandomUnderSampler'* permite amostrar dados heterogêneros conforme pode ser observado na Figura 20.

Figura 20 – Algortimo RandomUnderSampler com dados heterogêneos

```
>>> X_hetero = np.array([['xxx', 1, 1.0], ['yyy', 2, 2.0], ['zzz', 3, 3.0]],
...                     dtype=object)
>>> y_hetero = np.array([0, 0, 1])
>>> X_resampled, y_resampled = rus.fit_resample(X_hetero, y_hetero)
>>> print(X_resampled)
[['xxx' 1 1.0]
 ['zzz' 3 3.0]]
>>> print(y_resampled)
[0 1]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Também funcional para o *dataframe* do pandas, conforme pode ser observado na Figura 21.

Figura 21 – Algortimo RandomUnderSampler para pandas

```
>>> from sklearn.datasets import fetch_openml
>>> df_adult, y_adult = fetch_openml(
...     'adult', version=2, as_frame=True, return_X_y=True)
>>> df_adult.head()
>>> df_resampled, y_resampled = rus.fit_resample(df_adult, y_adult)
>>> df_resampled.head()
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O algoritmo *'NearMiss'* adiciona algumas regras heurísticas para selecionar amostras (MANI; ZHANG, 2003 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). *'NearMiss'* implementa 3 tipos diferentes de heurística que podem ser selecionados com o parâmetro *'version'*, conforme imagem Figura 22 (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Figura 22 – Algortimo NearMiss

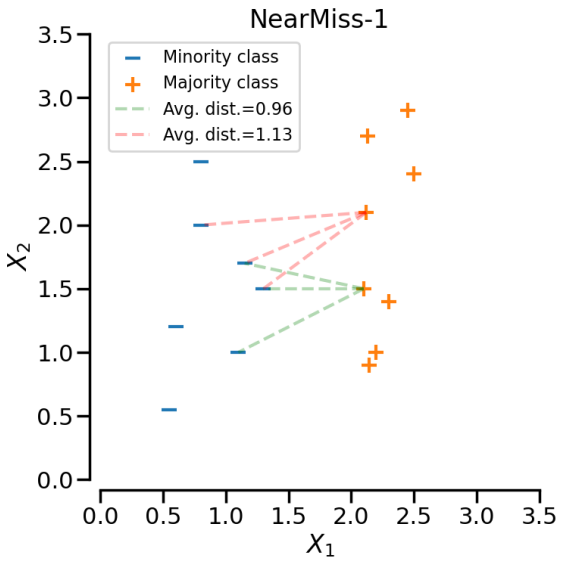
```
>>> from imblearn.under_sampling import NearMiss
>>> nm1 = NearMiss(version=1)
>>> X_resampled_nm1, y_resampled = nm1.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 64), (2, 64)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

As regras heurísticas *'NearMiss'* são baseadas no algoritmo dos vizinhos mais próximos. Portanto, os parâmetros *'n_neighbors'* e *'n_neighbors_ver3'* aceitam o classificador derivado de *'KNeighborsMixin'* do *scikit-learn* (PEDREGOSA et al., 2011). O primeiro parâmetro é utilizado para calcular a distância média aos vizinhos enquanto o segundo é utilizado para a pré-seleção das amostras de interesse (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Sejam amostras positivas as amostras pertencentes à classe de destino a serem subamostradas. A amostra negativa refere-se às amostras da classe minoritária (ou seja, a classe mais sub-representada). *NearMiss-1* seleciona as amostras positivas para as quais a distância média para as N amostras mais próximas da classe negativa é a menor, de acordo com o que pode ser observado na Figura 23. Nesta figura, os termos *'minority'*, *'majority'* e *'avg'*, podem ser compreendidos como 'minoritário', 'majoritário' e 'média', respectivamente.

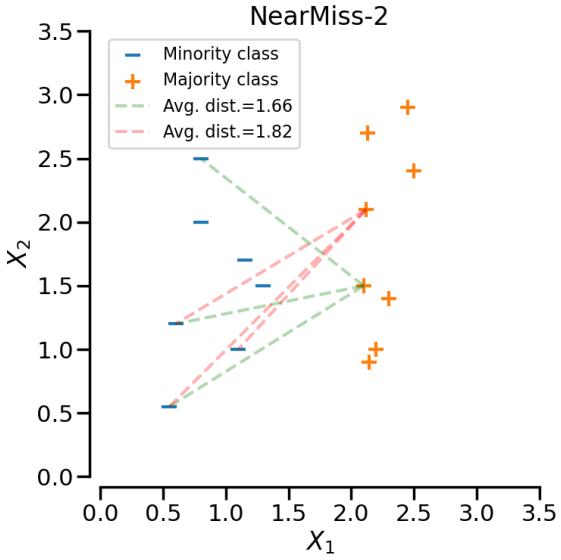
Figura 23 – NearMiss-1



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

NearMiss-2 (Figura 24) seleciona as amostras positivas para as quais a distância média para as N amostras mais distantes da classe negativa é a menor (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Figura 24 – NearMiss-2

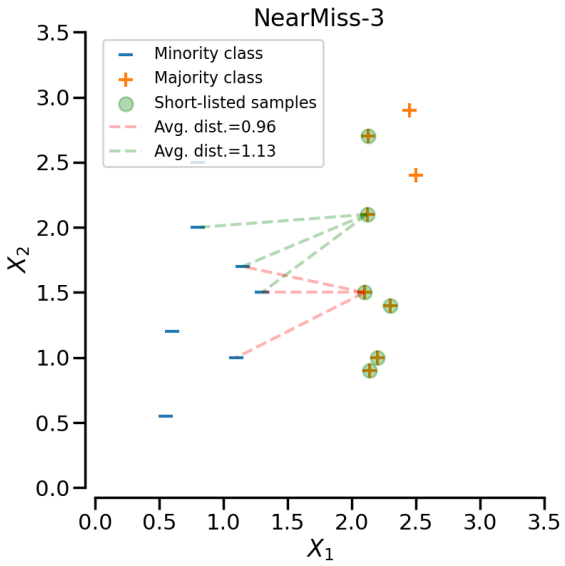


Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

NearMiss-3 (Figura 25) é um algoritmo de 2 passos. Primeiro, para cada amostra negativa, seus M vizinhos mais próximos serão mantidos. Então, as amostras positivas selecionadas são aquelas para as quais a distância média aos N vizinhos mais próximos é a maior

(LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

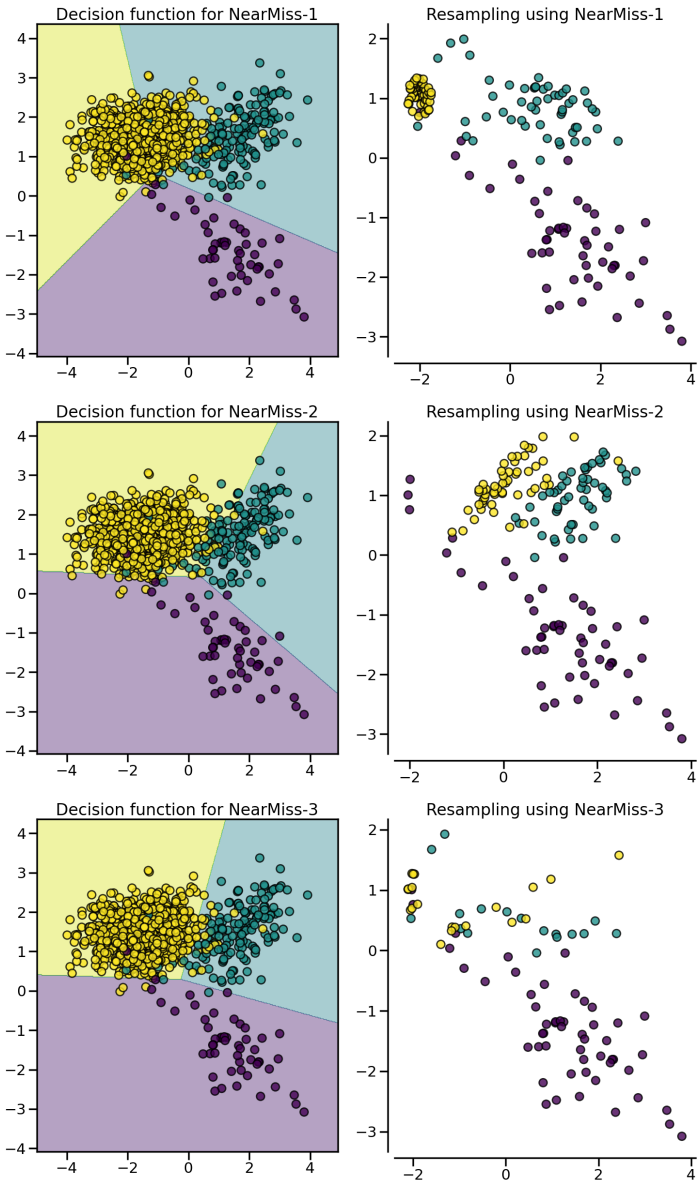
Figura 25 – NearMiss-3



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Ao subamostrar uma classe específica, o *NearMiss-1* pode ser alterado pela presença de ruído. Na verdade, isso implicará que as amostras da classe alvo serão selecionadas em torno dessas amostras, como é o caso da ilustração abaixo para a classe amarela. No entanto, no caso normal, as amostras próximas aos limites serão selecionadas. *NearMiss-2* não terá esse efeito, pois não foca nas amostras mais próximas, mas sim nas amostras mais distantes. Podemos imaginar que a presença de ruído também pode alterar a amostragem principalmente na presença de *outliers*marginais. *NearMiss-3* é provavelmente a versão que será menos afetada pelo ruído devido à seleção da amostra na primeira etapa. A Figura 26 ilustra este cenário.

Figura 26 – Reamostragem usando NearMiss-1, NearMiss-2 e NearMiss-3



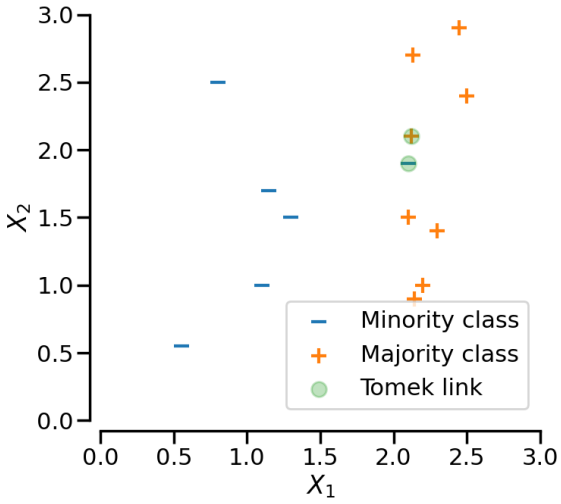
Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

As técnicas de subamostragem de limpeza não permitem especificar o número de amostras a ter em cada classe. Na verdade, cada algoritmo implementa uma heurística que limpará o conjunto de dados. *'TomekLinks'* detecta os chamados links de *Tomek* (IVAN, 1976 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Um *link de Tomek* entre duas amostras de classes diferentes x e y é definido de modo que, para qualquer amostra z:

$$d(x, y) < d(x, z) \text{ e } d(x, y) < d(y, z) \tag{2.50}$$

onde $d(.)$ é a distância entre as duas amostras. Em outras palavras, um *link de Tomek* existe se as duas amostras forem as vizinhas mais próximas uma da outra. Na Figura 27, um *link de Tomek* é ilustrado destacando as amostras de interesse em verde.

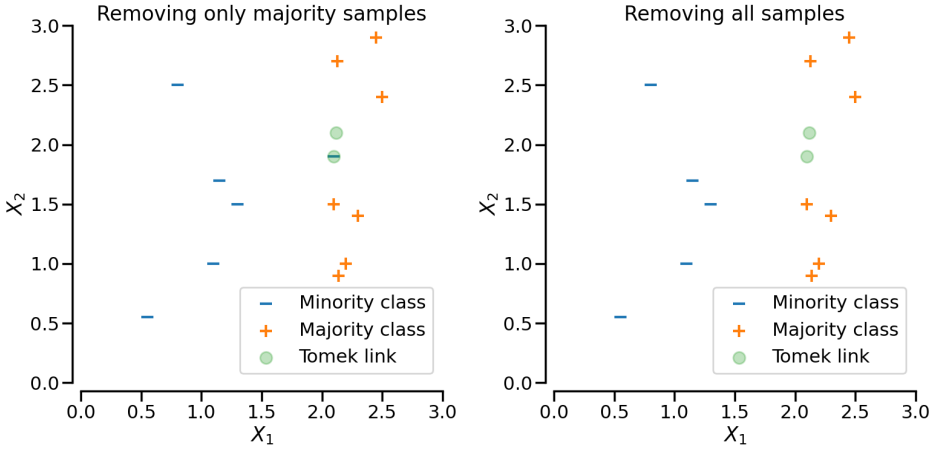
Figura 27 – Ilustração do *Link de Tomek*



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O parâmetro *'sampling_strategy'* controla qual amostra do *link* será removida. Por exemplo, o padrão (ou seja, *' sampling_strategy = 'auto' '*) removerá a amostra da classe majoritária. Ambas as amostras da classe majoritária e minoritária podem ser removidas definindo *'sampling_strategy'* como *'all'*. A Figura 28 ilustra esse comportamento.

Figura 28 – *Link de Tomek* removendo amostras da classe majoritária (esquerda) e de todas as amostras (direita)



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Já a técnica de balanceamento *'EditedNearestNeighbours'* aplica um algoritmo de vizinhos mais próximos e 'edita' o conjunto de dados removendo amostras que não concordam 'suficientemente' com sua vizinhança (WILSON, 1972 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Para cada amostra na classe a ser subamostrada, os vizinhos mais próximos são computados e se o critério de seleção não for atendido, a amostra é removida, conforme ilustrado na Figura 29.

Figura 29 – *EditedNearestNeighbours*

```
>>> sorted(Counter(y).items())
[(0, 64), (1, 262), (2, 4674)]
>>> from imblearn.under_sampling import EditedNearestNeighbours
>>> enn = EditedNearestNeighbours()
>>> X_resampled, y_resampled = enn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 213), (2, 4568)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Dois critérios de seleção estão atualmente disponíveis: (i) a maioria (ie, *'kind_sel = 'mode'*) ou (ii) todos (ie, *'kind_sel = 'all'*) os vizinhos mais próximos devem pertencer à mesma classe que a amostra inspecionada para mantê-la no conjunto de dados. Assim, implica que *' kind_sel = 'all' '* será menos conservador do que *' kind_sel = 'mode' '*, e mais amostras serão excluídas na primeira estratégia do que na última, conforme Figura 30.

Figura 30 – *EditedNearestNeighbours* com critérios

```
>>> enn = EditedNearestNeighbours(kind_sel="all")
>>> X_resampled, y_resampled = enn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 213), (2, 4568)]
>>> enn = EditedNearestNeighbours(kind_sel="mode")
>>> X_resampled, y_resampled = enn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 234), (2, 4666)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O parâmetro *'n_neighbors permite'* fornecer um classificador subclassificado de *'KNeighborsMixin'* do *scikit-learn* para encontrar os vizinhos mais próximos e tomar a decisão de manter uma determinada amostra ou não (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

O método de balanceamento *undersampling 'RepeatedEditedNearestNeighbours'* é uma variação do *'EditedNearestNeighbours'* repetindo o algoritmo mais vezes (TOMEK, 1976 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Geralmente repetir o algoritmo exclui mais dados. O algoritmo é demonstrado na Figura 31.

Figura 31 – *RepeatedEditedNearestNeighbours*

```
>>> from imblearn.under_sampling import RepeatedEditedNearestNeighbours
>>> renn = RepeatedEditedNearestNeighbours()
>>> X_resampled, y_resampled = renn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 208), (2, 4551)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

'AllKNN' difere do anterior do *'RepeatedEditedNearestNeighbours'* pois o número de vizinhos do algoritmo interno de vizinhos mais próximos é aumentado a cada iteração (TOMEK, 1976 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). O algoritmo é demonstrado na Figura 32.

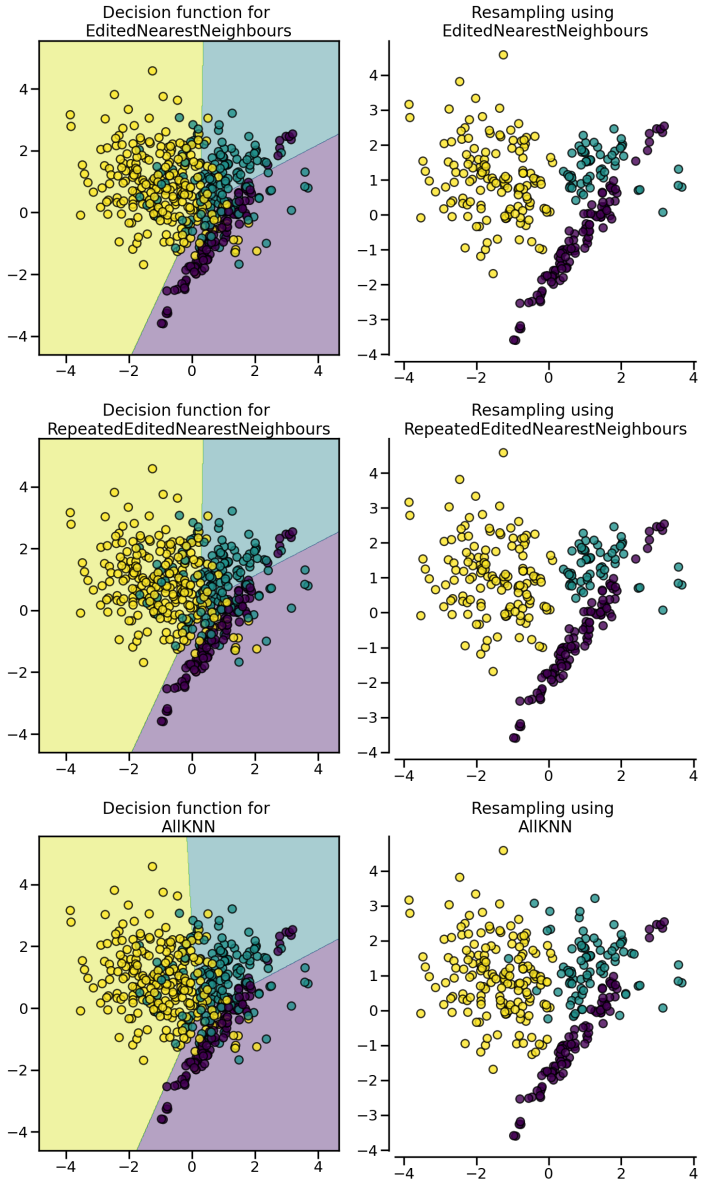
Figura 32 – *AllKNN*

```
>>> from imblearn.under_sampling import AllKNN
>>> allknn = AllKNN()
>>> X_resampled, y_resampled = allknn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 220), (2, 4601)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Na Figura 33 pode-se ver que os três algoritmos têm impacto semelhante ao limpar amostras ruidosas próximas aos limites das classes (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Figura 33 – Reamostragem usando *AllKNN*, *RepeatedEditedNearestNeighbours* e *EditedNearestNeighbours*



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O *'CondensedNearestNeighbour'* usa uma regra de 1 vizinho mais próximo para decidir

iterativamente se uma amostra deve ser removida ou não (HART, 1968 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). O algoritmo está sendo executado da seguinte forma:

- Obtenha todas as amostras minoritárias em um conjunto C;
- Adicione uma amostra da classe de destino (classe a ser subamostrada) em C e todas as outras amostras desta classe em um conjunto S;
- Percorra o conjunto S, amostra por amostra e classifique cada amostra usando uma regra de 1 vizinho mais próximo;
- Se a amostra for classificada incorretamente, adicione-a C, caso contrário, não faça nada;
- Reiterar em S até que não haja amostras a serem adicionadas.

Seu uso pode ser implementado conforme demonstrado na Figura 34.

Figura 34 – *CondensedNearestNeighbour*

```
>>> from imblearn.under_sampling import CondensedNearestNeighbour
>>> cnn = CondensedNearestNeighbour(random_state=0)
>>> X_resampled, y_resampled = cnn.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 24), (2, 115)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

No entanto, conforme ilustrado na Figura 37, *'CondensedNearestNeighbour* é sensível ao ruído e adicionará amostras ruidosas.

Ao contrário, *'OneSidedSelection'* usará *'TomekLinks'* para remover amostras ruidosas (HART, 1968 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Além disso, a regra do um vizinho mais próximo é aplicada a todas as amostras e aquela que for mal classificada será adicionada ao conjunto C. Nenhuma iteração no conjunto S ocorrerá. A classe pode ser usada como demonstrado na Figura 35.

Figura 35 – *OneSidedSelection*

```
>>> from imblearn.under_sampling import OneSidedSelection
>>> oss = OneSidedSelection(random_state=0)
>>> X_resampled, y_resampled = oss.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 174), (2, 4404)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Para definir o número de sementes para colocar no conjunto C originalmente, definindo o parâmetro *'n_seeds_S'*. *'NeighbourhoodCleaningRule'* se concentrará na limpeza dos dados, em vez de condensá-los (LAURIKKALA, 2001 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). Portanto, será utilizada a união de amostras a serem rejeitadas entre os *'EditedNearestNeighbours'* e a saída de um classificador de três vizinhos mais próximos. A classe pode ser usada como demonstrado na Figura 36.

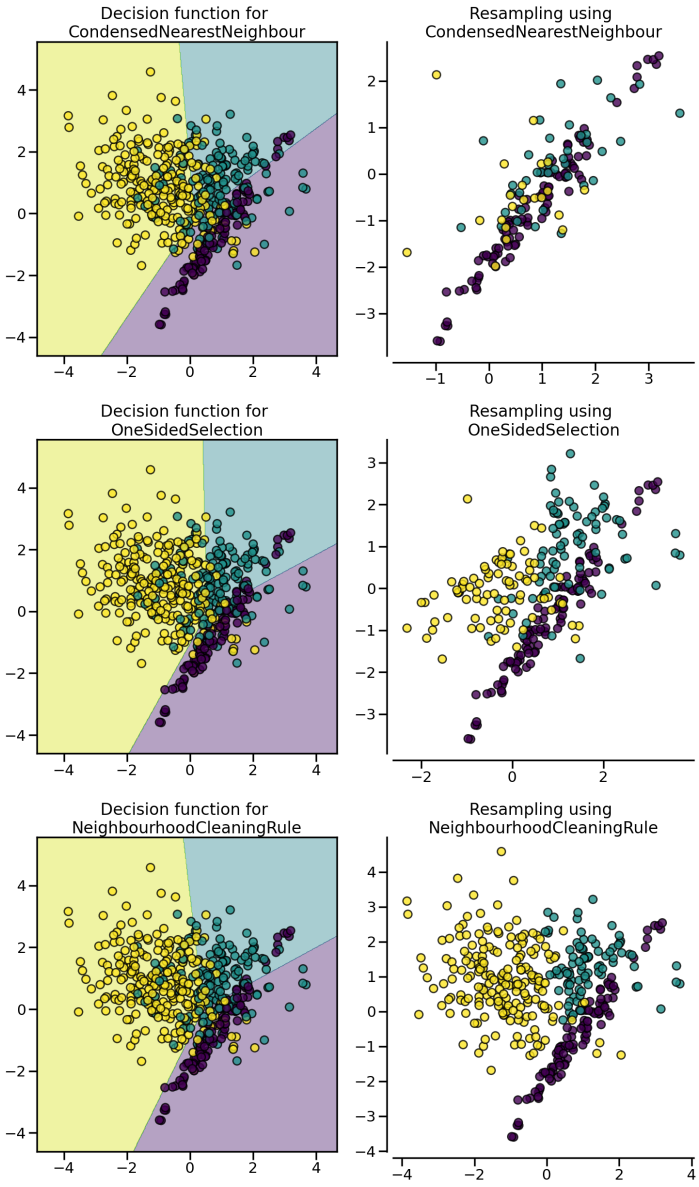
Figura 36 – *NeighbourhoodCleaningRule*

```
>>> from imblearn.under_sampling import NeighbourhoodCleaningRule
>>> ncr = NeighbourhoodCleaningRule()
>>> X_resampled, y_resampled = ncr.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 234), (2, 4666)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

O resultado do algoritmo acima pode ser observado na Figura 37.

Figura 37 – Reamostragem usando *'CondensedNearestNeighbour*, *OneSidedSelection* e *NeighbourhoodCleaningRule*



Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

'InstanceHardnessThreshold' é um algoritmo específico no qual um classificador é treinado nos dados e as amostras com probabilidades menores são removidas (SMITH; MARTI-

NEZ; GIRAUD-CARRIER, 2014 apud LEMAÎTRE; NOGUEIRA; ARIDAS, 2017). A classe pode ser usada como demonstrado na Figura 38.

Figura 38 – *InstanceHardnessThreshold*

```
>>> from sklearn.linear_model import LogisticRegression
>>> from imblearn.under_sampling import InstanceHardnessThreshold
>>> iht = InstanceHardnessThreshold(random_state=0,
...                                 estimator=LogisticRegression(
...                                     solver='lbfgs', multi_class='auto'))
>>> X_resampled, y_resampled = iht.fit_resample(X, y)
>>> print(sorted(Counter(y_resampled).items()))
[(0, 64), (1, 64), (2, 64)]
```

Fonte: (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017)

Esta classe tem dois parâmetros importantes. *'estimator'* aceitará qualquer classificador *scikit-learn* que tenha um método *'predict_proba'*. O treinamento do classificador é realizado usando uma validação cruzada e o parâmetro *'cv'* pode definir o número de dobras a serem usadas (LEMAÎTRE; NOGUEIRA; ARIDAS, 2017).

Desse modo, o algoritmo *'RandomUnderSampler'* por ser um método de subamostragem que visa resolver o problema de desbalanceamento reduzindo aleatoriamente o número de exemplos da classe majoritária, para que fique mais equilibrado em relação à classe minoritária, foi o algoritmo *undersampling* escolhido para ser utilizado neste trabalho.

2.7 Trabalhos correlatos

Esta seção descreve as percepções inferidas com base no estudo da literatura, no que diz respeito ao uso do aprendizado de máquina, bem como na utilização de algoritmos classificadores e métricas aplicados ao problema de risco de crédito.

Ao realizar uma revisão sistemática de literatura sobre aprendizado de máquina em risco de crédito e aplicar os algoritmos *Support Vector Machine*, *Decision Tree*, *Bagging*, *AdaBoost* e *Random Forest*, Aniceto (2016) obteve o melhor *Area Under the Curve* (AUC) para os métodos de classificadores *ensemble*: *AdaBoost*, *Random Forest* e *Bagging*, respectivamente. A base de dados utilizada possui diferentes quantidades de amostras contendo informações de uma instituição financeira brasileira de grande porte atuante no mercado de crédito, com dados históricos dos clientes do período de setembro de 2007 a janeiro de 2010 com prazos de contratação de 24 meses.

Os algoritmos *Support Vector Machine* (SVM) apresentaram melhor sensibilidade, porém, baixa especificidade. Assim sendo, os 3 algoritmos citados anteriormente também obtiveram a melhor especificidade: *AdaBoost* foi o melhor, seguido por *Bagging* e *Random Forest*.

Em seu trabalho Batista (2018) aplicou os algoritmos de aprendizado de máquina supervisionado em um problema real de classificação com o objetivo de classificar os clientes de um banco como bons ou maus pagadores. Os dados foram obtidos a partir da base de dados intitulada "*Default Payment Nex Month*", disponibilizada no repositório da *UCI Machine Learning Repository* (ASUNCION; NEWMAN, 2007). Os algoritmos utilizados foram: *Logistic Regression*, classificadores bayesianos, *K-Nearest Neighbors*, *Random Forest* e redes neu-

rais. Analisando os resultados obtidos, ao utilizar as redes neurais é apresentado menos classificações incorretas se comparado aos demais métodos, porém, a taxa de erro de falsos negativos é alta, ou seja, um mau pagador pode ser considerado como um bom pagador com mais frequência se comparado aos outros métodos avaliados. *Random Forest* foi considerado o melhor quando se considera o valor da área sob a curva ROC (*Receiver Operating Characteristic*), porém, após análise, o autor considera o método *Logistic Regression* como a melhor relação custo-benefício, pois apresenta um bom preditivo e comete erros com menos frequência.

Fernandes (2019) relata em seu trabalho que bancos e organizações financeiras que lidam com o empréstimo de crédito têm como chave para o sucesso os modelos de avaliação de crédito para classificar seus clientes em perspectiva como prováveis adimplentes ou inadimplentes, porém, ressalta que é possível tomar essas decisões substanciais com mais precisão através do suporte de modelos desenvolvidos a partir do aprendizado de máquina usando fundamentalmente a sua tarefa de classificação. Para isso, com o objetivo de realizar uma avaliação de algoritmos classificatórios consolidadas na literatura, foram realizados experimentos conduzidos em três conjunto de dados de crédito disponíveis publicamente no repositório de dados da *UCI Machine Learning Repository* (ASUNCION; NEWMAN, 2007) e comparado a acurácia média dos algoritmos. Utilizando os classificadores *Decision Tree*, *Random Forest*, *Support Vector Machine*, *Logistic Regression* e redes neurais artificiais, ao final do experimento, os resultados obtidos mostram que o classificador *Random Forest* se sobressaiu às demais técnicas por apresentar a maior acurácia média e por ser a mais estável para as três bases de dados estudadas, comportamento este, similar ao observado por Aniceto (2016) e Batista (2018). A rede neural apresentou desempenho inconsistente e *Logistic Regression* apresentou bom desempenho.

Realizado estudo, de maneira similar a Batista (2018), com objetivo de prever quais clientes são bons ou maus pagadores e realizar uma análise comparativa entre as técnicas de previsão de inadimplência para avaliação de risco de transações com cartão de crédito, *Logistic Regression*, *Support Vector Machine* e redes neurais, Sinhorigno (2007) obteve melhor desempenho no modelo neural multicamada de arquitetura *feed-foward* com taxa de acerto de 78,46% e teste de qualidade de ajuste, conhecido como teste de Kolmogorov-Smirnov (KS) de 55,5%. *Logistic Regression* apresentou o segundo melhor resultado, com taxa de acerto de 76,91% e KS de 54,26%. O que, se comparado aos resultados de Fernandes (2019), temos uma divergência de conclusões para as redes neurais e resultados similares para a técnica de *Logistic Regression*.

No estudo comparativo entre metodologias de aprendizado de máquina e híbridas aplicadas a risco de crédito, Castro (2019) comparou as técnicas *Logistic Regression*, *Support Vector Machine*, *Random Forest*, modelos híbridos, e *Gradient Boosting*. As técnicas foram aplicadas às informações dos brasileiros provenientes da empresa Serasa Experian. A comparação dos modelos via métricas de performance AUC, KS e Taxa de acerto, indicou o *Gradient Boosting* como o melhor.

O trabalho realizado por Lopes (2022) possui o objetivo de comparar os modelos *k*-

Nearest Neighbors, *Decision Tree*, *Random Forest*, *Logistic Regression*, *Support Vector Machines*, *Multilayer Perceptron*, *XGBoost*, *AdaBoost* e um método *Ensemble Stack*, a fim de encontrar o modelo com melhor desempenho na análise de crédito. Para isso, foram utilizadas três bases públicas, que estão disponíveis na *UCI Machine Learning Repository* (ASUNCION; NEWMAN, 2007), sendo elas: *German Credit Data*, *Australian Credit Approval* e *Default of Credit Card Clients Data Set*. Foi observado que o método *Ensemble Stack* obteve maior acurácia com a média de 81,41%, *XGBoost* ocupou a segunda posição com 80,87%, seguido por *Logistic Regression* com 80,48% de acurácia média. No que diz respeito a precisão, temos *Ensemble Stack* (71,40%), *Logistic Regression* (70,92%) e *XGBoost* (69,51%) com os resultados mais precisos. Para a métrica *F1 Score* Lopes obteve os melhores resultados com o *XGBoost* (61,65%), *Ensemble Stack* (61,32%) e com a rede neural (61,14%).

Realizando uma comparação entre a regressão logística clássica e dois métodos de aprendizado de máquina para *credit scoring*, *Random Forest* e *XGBoost*, visando identificar qual apresenta melhor desempenho na previsão de inadimplência, para uma amostra composta por dados de micro, pequenas e médias empresas com atuação em todo o território brasileiro, Coelho, Amorim e Camargos (2021), com base nos resultados do estudo realizado, evidenciaram que os métodos de aprendizado de máquina apresentaram melhor desempenho se compara a análise convencional, baseada na *Logistic Regression*. O método obteve 30,36% no teste de Kolmogorov-Smirnov (KS) e acurácia (ACC) de 63,02%. O *Random Forest* obteve KS de 47,00% e ACC de 66,81%. Já o *XGBoost* obteve KS de 57,12% e ACC de 72,04%, obtendo assim a maior capacidade de previsão de inadimplência.

Por meio de um processo de seleção de artigos cientificamente relevantes no campo de pesquisa escolhido por Marra (2019), obteve-se uma amostra final de 168 estudos considerados hábeis a nortear os rumos do uso da aprendizagem de máquinas aplicados às finanças. A partir dessa amostra de estudos, provenientes das bases de dados: *Web of Science*, *SCOPUS*, *Science Direct* e *Emerald*, realizou-se uma análise mais profunda por meio de leituras realizadas pelo autor e descobriu-se que os algoritmos de aprendizagem de máquina estão sendo explorados, aprimorados e levados ao extremo para detectar combinações sutis e melhor descrever o risco de crédito. Da mesma forma que Lopes (2022), bem como, Coelho, Amorim e Camargos (2021), Marra (2019) também concluiu que o modelo *XGBoost* (96,05% de acurácia) foi capaz de apresentar melhores resultados, quando comparados com *Random Forest* (95,10%) e *Logistic Regression* (65,12%).

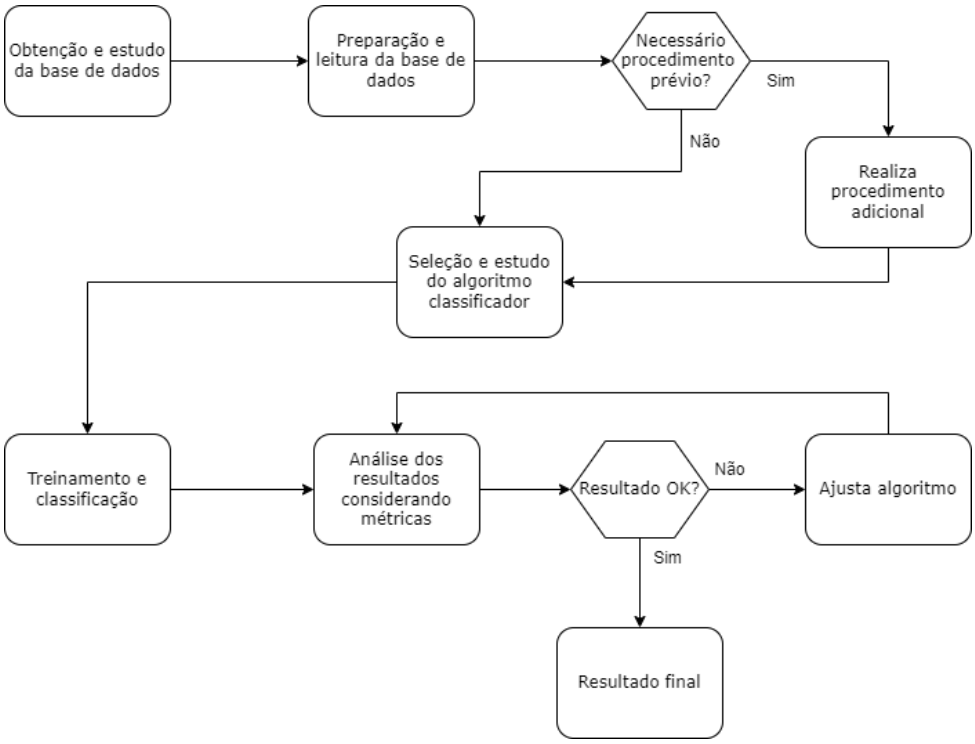
Destarte, segundo a literatura presente nos trabalhos de Aniceto (2016), Batista (2018), Fernandes (2019), Sinhorigno (2007), Castro (2019), Lopes (2022), Coelho, Amorim e Camargos (2021) e Marra (2019), as técnicas *XGBoost, Random Forest, Gradient Boosting, AdaBoost* e *Logistic Regression* se destacam devido a constante presença em trabalhos da área e principalmente devido aos resultados observados pelos autores apresentados acima.

3 Materiais e Métodos

O presente trabalho pode ser classificado como uma pesquisa exploratória, conforme definido por Wazlawick (2009), pois busca utilizar o aprendizado de máquina para classificar usuários inadimplentes de uma empresa utilizando algoritmos classificadores. Tais classificadores foram selecionados através de uma revisão de literatura a fim de treinar a base de dados selecionada de modo a criar um modelo a ser submetido em uma base de teste e, utilizando métricas, ter seus resultados avaliados e comparados.

Dessa forma, pode também ser considerado um estudo de caso, conforme definido por Yin (2015), pois busca aplicar o aprendizado de máquina em uma base de dados específica, a fim de obter resultados que possam ser utilizados para a tomada de decisão de uma empresa, realizando uma investigação empírica compreendendo um método abrangente, com a lógica do planejamento, da coleta e da análise de dados (VENTURA, 2007). Para isso, faz-se necessário algumas etapas, conforme pode ser observado no fluxograma presente na Figura 39.

Figura 39 – Fluxograma das etapas desenvolvidas neste trabalho



Fonte: elaborada pelo autor

3.1 Obtenção e estudo da base de dados

A base de dados e principal material utilizado neste trabalho, está disponível na plataforma *Kaggle* (KAGGLE, 2019), através de uma competição AMEX intitulada "*AmExpert 2021*

CODELAB – Machine Learning Hackathon" [1] . Ela contém informações relevantes sobre os usuários de cartão de crédito de uma empresa, como: nome, idade, sexo, se possui carro, se possui casa, quantidade de filhos, renda, profissão, limite atual do cartão, porcentagem do limite utilizado, pontuação de crédito (*score*), número de inadimplência anteriores, se ficou inadimplente nos últimos 6 meses, entre outras. São disponibilizados em formato *csv*, uma base de treino (com 190 atributos) contendo 5.531.451 registros (*'train_data.csv'*) com seus rótulos presentes em um arquivo a parte (*'train_labels.csv'*) e outros arquivos adicionais que não foram utilizados (*'test_data.csv'* e *'sample_submission.csv'*), totalizando mais de 45 GB de dados (KAGGLE, 2022).

Para que seja possível utilizar esses dados para treinar os algoritmos classificadores da melhor maneira possível foi realizado uma análise empírica dos dados, a fim de verificar a necessidade de realizar um pré-processamento que irá anteceder o treinamento do algoritmo classificador, como por exemplo, realizar o balanceamento da base de dados ou ajustes (tratamento) em possíveis valores inválidos.

3.2 Preparação e leitura da base de dados

Após estudo da base de dados realizado na etapa anterior, foi realizado um procedimento adicional (pré-processamento), de modo a garantir que os registros estejam balanceados, ou seja, que não tenham grandes quantidades de um determinado rótulo em relação aos demais. Além disso, é importante que também não possua valores inválidos, como valores nulos por exemplo. Neste caso, foi utilizado o método "fillna" para tratar estes valores inexistentes (MCKINNEY, 2021).

A base de dados foi lida através de um algoritmo desenvolvido. Foi realizada a leitura utilizando a técnica *k-fold* que realizou a separação em base de treino e teste conforme explicado na seção seção 2.4. Se tratando de uma extensa base de dados (com muitos registros contendo muitos atributos) e considerando memória RAM disponível, a leitura foi limitada.

3.3 Escolha e estudo dos algoritmos classificadores

Os algoritmos classificadores foram escolhidos através da pesquisa e estudo de trabalhos similares ao objetivo deste trabalho de modo a identificar técnicas pertinentes a base de dados escolhida. Os classificadores que obtiveram as melhores métricas (acurácia, precisão, *recall*, *F1-score* - conforme apresentado na seção 2.3) foram utilizados para o treinamento.

Após seleção de algoritmos classificadores candidatos a serem utilizados neste trabalho, realizou-se um estudo da literatura de modo a obter conhecimentos teóricos, a fim de compreender o funcionamento; e prático, com o objetivo de obter sucesso em sua implementação. Por fim, após este estudo, três algoritmos classificadores foram utilizados.

[1] https://www.kaggle.com/datasets/hotsonhonet/amex-competition

3.4 Treinamento e classificação dos usuários

Nesta etapa implementou-se os algoritmos classificadores conforme estudo realizado em etapa anterior. Realizou-se o treinamento destes algoritmos classificadores utilizando a base de dados selecionada na seção 3.1.

Um programa codificado na linguagem Python analisou as entradas presentes no arquivo da base de dados e utilizou os algoritmos classificadores selecionados na etapa apresentada na seção 3.3 para realizar a previsão da saída, ou seja, definir se com as características apresentadas, o usuário analisado está ou não inadimplente.

3.5 Análise dos resultados

Utilizando as métricas apresentadas na seção 2.3, os modelos treinados na etapa anterior foram submetidos a uma base de teste, para que seja possível avaliar os resultados obtidos. Esta base de teste contem uma parcela de registros da base de dados original (seção 2.4), e a fim de tornar possível comparar os resultados obtidos com os desejados, a base de teste teve seus rótulos previstos pelo algoritmo classificador e ao final do processo, foram comparados com os rótulos dos mesmos registros da base original. A técnica de *k-fold* considerou 10 partições, sendo que 8 foram utilizadas para treinamento e 2 para teste, ou seja, para a base com um milhão de registros, foram utilizados 800 mil registros para treinamento e 200 mil para teste. Tal proporção pode ser observada na literatura, conforme apresentado por Yadav 2016.

Com o êxito deste processo, os resultados dos algoritmos classificadores e métricas utilizadas foram comparados.

4 Desenvolvimento

Este capítulo apresenta a base de dados escolhida, os procedimentos desenvolvidos para efetuar uma seleção de três algoritmos classificadores a partir do processo de pesquisa e seleção de trabalhos relevantes, bem como os procedimentos necessários a fim de realizar o treinamento da base de dados utilizando distintas técnicas. A partir do estudo teórico e prático efetuado sobre tais algoritmos, é posteriormente analisado alguns critérios para realizar a seleção deles. Em seguida é descrito como os classificadores selecionados foram treinados e utilizados para classificar a base de dados. Por fim, no capítulo sucessor a este, os resultados obtidos a partir dos procedimentos realizados são analisados e comparados.

4.1 Base de dados e algoritmos classificadores

Conforme estudo realizado na seção 2.7, foi possível previamente identificar alguns algoritmos classificadores adequados para a base de dados deste trabalho que podem ser utilizadas para classificar os usuários de cartão de crédito de uma empresa. Entre eles, se destacam: *XGBoost, Random Forest, Gradient Boosting, AdaBoost* e *Logistic Regression*.

Devido os resultados apresentados, conforme pode ser observado na Tabela 1, infere-se que os algoritmos classificadores *XGBoost, Random Forest e Logistic Regression* apresentaram os melhores valores para as métricas de *accuracy*, *sensitivity*, curva ROC, KS e *precision*, de maneira mais frequente e portanto acredita-se que são os mais indicados para o propósito deste trabalho. *XGBoost* foi considerado o melhor (em verde escuro) ou o segundo melhor classificador (em verde claro) por todos os autores que o utilizou.

Assim sendo, selecionados três dos cinco algoritmos que se destacaram, realizou-se um estudo teórico através da leitura de seu acervo documental, e prático, observando exemplos de utilização e realizando testes de implementação sobre cada um deles, a fim de compreender sobre o processo de instalação, funcionamento e como podem ser utilizados para classificar a base de dados selecionada para este trabalho.

Tabela 1 – Comparativo entre os algoritmos classificadores apresentados na seção 2.7

Autor(es)	Gradient Boosting	XGBoost	KNN	Logistic Regression	Decision Tree	Random Forest	SVM	RNA	Bagging	AdaBoost	Bayes	Ensemble Stack
Aniceto (2016)												
Batista (2018)												
Fernandes (2019)												
Sinhorigno (2007)												
Castro (2019)												
Lopes (2022)												
Marra (2019)												
Coelho et al. (2021)												

Fonte: elaborada pelo autor

4.2 Leitura e análise da base de dados

De posse dos arquivos obtidos na plataforma *Kaggle*, o primeiro passo a fim de possibilitar a utilização dos dados é realizar a leitura dos arquivos. Para isso, foi utilizado a linguagem de programação *Python* e a biblioteca *Pandas* para realizar a leitura dos arquivos e a criação de um *DataFrame*. O *DataFrame* é uma estrutura de dados bidimensional, com linhas e colunas, remetendo a uma planilha que pode ser criado a partir de arquivos, página *web* ou dados gerados através de códigos, de modo a tornar possível realizar manipulações nos dados de maneira simples (MAIA et al., 2019). Já a biblioteca *Pandas* é uma biblioteca de código aberto que fornece estruturas de dados e ferramentas de análise e manipulação de dados de alto desempenho para a linguagem de programação *Python* (CHEN, 2018).

Analisando o DataFrame, foi possível inferir que ele possui 531.4515. registros, 190 colunas e possui dados do tipo *float*, alguns valores inteiros, datas e algumas *strings*. Uma parcela da base de dados pode ser observado na Figura 40.

Figura 40 – Primeiros registros do *DataFrame*

	customer_ID	S_2	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	...	D_136	D_137	D_138	D_139	D_140	D_141
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938469	0.001733	0.008724	1.006838	0.009228	0.124035	0.008771	0.004709	...	NaN	NaN	NaN	0.002427	0.003706	0.003818
1	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	0.936665	0.005775	0.004923	1.000653	0.006151	0.126750	0.000798	0.002714	...	NaN	NaN	NaN	0.003954	0.003167	0.005032
2	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	0.954180	0.091505	0.021655	1.009672	0.006815	0.123977	0.007598	0.009423	...	NaN	NaN	NaN	0.003269	0.007329	0.000427
3	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-06-13	0.960384	0.002455	0.013683	1.002700	0.001373	0.117169	0.000685	0.005531	...	NaN	NaN	NaN	0.006117	0.004516	0.003200
4	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-07-16	0.947248	0.002483	0.015193	1.000727	0.007605	0.117325	0.004653	0.009312	...	NaN	NaN	NaN	0.003671	0.004946	0.008889

Fonte: elaborada pelo autor

Nota-se que, os dados presentes no arquivo disponibilizado pela plataforma estão anonimizados, dessa forma, não é possível identificar o que cada coluna representa. Porém, com base nas informações presentes na página da competição onde foi obtida a base de dados, temos que as colunas:

- D_* = Variáveis de inadimplência;
- S_* = Variáveis de despesa;
- P_* = Variáveis de pagamento;
- B_* = Variáveis de equilíbrio;
- R_* = Variáveis de risco.

A Figura 41 a seguir, apresenta algumas estatísticas do *DataFrame*. Nela é possível observar a quantidade de registros presentes em algumas colunas (*count*), a média (*mean*), o desvio padrão (*std*), o valor mínimo (*min*), o valor máximo (*max*) e os valores dos quartis (*25%*, *50%* e *75%*).

Figura 41 – Resumo das estatísticas do Dataset

	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	D_42	D_43	...	D_136	D_137
count	5.485466e+06	5.531451e+06	5.531451e+06	5.529435e+06	5.531451e+06	4.510907e+06	5.529435e+06	5.529435e+06	791314.000000	3.873055e+06	...	1.946990e+05	1.946990e+05
mean	6.563340e-01	1.531172e-01	1.240100e-01	6.214887e-01	7.880270e-02	2.258455e-01	5.978469e-02	1.325389e-01	0.184974	1.546841e-01	...	2.427725e-01	1.424409e-02
std	2.446494e-01	2.700709e-01	2.119869e-01	4.014877e-01	2.263971e-01	1.933475e-01	2.025443e-01	2.349929e-01	0.228185	2.133977e-01	...	2.101320e-01	9.571115e-02
min	-4.589548e-01	5.026190e-09	-7.588799e+00	9.192280e-09	1.534223e-09	-6.271320e-01	5.566545e-10	6.285293e-09	-0.000454	1.154550e-07	...	6.316773e-08	1.078787e-08
25%	4.803307e-01	4.528464e-03	8.863645e-03	1.053313e-01	2.895934e-03	1.272588e-01	2.873244e-03	5.227570e-03	0.037516	4.227546e-02	...	9.314305e-03	2.532470e-03
50%	6.942950e-01	9.056902e-03	3.132968e-02	8.143328e-01	5.782230e-03	1.639082e-01	5.746725e-03	9.777230e-03	0.120519	8.851245e-02	...	2.539468e-01	5.069830e-03
75%	8.648159e-01	2.366407e-01	1.259019e-01	1.002403e+00	8.660590e-03	2.581017e-01	8.615665e-03	1.550507e-01	0.250869	1.843206e-01	...	2.582450e-01	7.573434e-03
max	1.010000e+00	5.389619e+00	1.324060e+00	1.010000e+00	3.256284e+00	5.482888e+00	8.988807e+00	1.625262e+00	4.191119	1.011162e+01	...	1.759910e+00	1.009998e+00

Fonte: elaborada pelo autor

Neste atual estágio, o *DataFrame* não possui rótulos pois eles estão presentes em outro arquivo, intitulado ”train_labels”. Desse modo, foi preciso realizar a leitura do outro arquivo e em seguida realizar a união dos dois *DataFrames* utilizando a coluna ”customer_ID” como chave identificadora. Assim, o campo de identificação do cliente não possui nenhum valor relevante para o aprendizado de máquina e sua coluna foi então removida do *DataFrame*.

Após a inclusão dos rótulos (*target*) no *DataFrame*, foi possível verificar que o conjunto de dados estava desbalanceado, pois a classe 0 possui uma quantidade muito maior de registros quando comparado a classe 1. A Figura 42 apresenta a distribuição das classes, onde 0 representa os adimplentes (classe negativa) e 1 os inadimplentes (classe positiva).

Figura 42 – Situação do desbalanceamento da base

```
[22]  df['target'].value_counts()

...   0    4153582
      1    1377869
      Name: target, dtype: int64
```

Fonte: elaborada pelo autor

4.3 Tratamento da base de dados

Após realizar a leitura dos dados, foi possível inferir que a base possui alguns dados faltantes, que são representados por valores nulos. Além disso, a base de dados possui duas colunas (”D_63” e ”D_64”) que são categóricas, contendo textos como ”CO” e ”O” e também necessitam de um tratativa antes de realizar o treinamento da base. Algo semelhante acontece com a coluna ”S_2” que possui datas que se repetem.

Assim, para que o resultado final não fosse prejudicado por estas características, foi necessário realizar ajustes na base de dados. Para tratar a presença de valores nulos, foi utilizado o método ”fillna()” que substitui os valores nulos por 0. Em seguida, foi aplicada a técnica de *One-Hot Encoding* para transformar as variáveis categóricas em variáveis numéricas de forma a não influenciar de forma equivocada alguns algoritmos. A conversão funciona de

forma simples: para cada categoria em uma determinada coluna, é criada uma nova coluna onde o valor será 1 quando a linha tiver aquele valor para a categoria ou 0 caso contrário (PINTO, 2021 apud JIANG; LIN; RAGHAVAN, 2020).

Para a coluna ”S_2” que apresenta datas que se repetem (totalizando 396 valores únicos frente aos 5,5 milhões de dados), foi aplicada a técnica de *Label Encoding* para transformar as datas em valores numéricos. Esta técnica é utilizada com o intuito de converter as classes categóricas em valores numéricos, onde cada classe é mapeada para um valor inteiro de modo a facilitar o processamento dos dados (RASCHKA; MIRJALILI, 2017 apud ARAUJO, 2020).

Realizando um comparativo com a base original, através da aferição de esquemas, nota-se que os problemas citados anteriormente foram solucionados. Assim sendo, a base de dados está pronta para ser utilizada nos algoritmos de classificação. Foi realizada então o salvamento em um novo arquivo *csv* com a base de dados preparada que pode ser observada na Figura 43.

Figura 43 – Situação da base de dados após os tratamentos

	S_2	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	D_42	...	D_63_CR	D_63_XL	D_63_XM	D_63_XZ	D_64_0	D_64_-1
0	8	0.938469	0.001733	0.008724	1.006838	0.009228	0.124035	0.008771	0.004709	0.0	...	1	0	0	0	0	0
1	37	0.936665	0.005775	0.004923	1.000653	0.006151	0.126750	0.000798	0.002714	0.0	...	1	0	0	0	0	0
2	88	0.954180	0.091505	0.021655	1.009672	0.006815	0.123977	0.007598	0.009423	0.0	...	1	0	0	0	0	0
3	104	0.960384	0.002455	0.013683	1.002700	0.001373	0.117169	0.000685	0.005531	0.0	...	1	0	0	0	0	0
4	137	0.947248	0.002483	0.015193	1.000727	0.007605	0.117325	0.004653	0.009312	0.0	...	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	318	0.373642	0.118765	0.049370	0.815200	0.009317	0.148005	0.876761	0.049915	0.0	...	1	0	0	0	0	0
9995	362	0.331150	1.472028	0.049780	0.187973	0.000417	0.143625	0.955153	0.099672	0.0	...	1	0	0	0	0	0
9996	383	0.361060	2.065567	0.048313	0.194930	0.006775	0.144908	1.252287	0.118775	0.0	...	1	0	0	0	0	0
9997	27	0.708199	0.357349	0.013953	1.002573	0.003448	0.110258	0.170252	0.040633	0.0	...	0	0	0	0	0	0
9998	45	0.692401	0.009380	0.018337	0.588057	0.001596	0.108820	0.177197	0.031383	0.0	...	0	0	0	0	0	0

Fonte: elaborada pelo autor

4.4 Classificação

Fazendo uso do Pandas, a base de dados, preparada na seção 4.3, foi lida e a coluna contendo os rótulos (*target*) foi separada das demais, de modo a ter os dados do *dataset* (X) e os rótulos (y), que foram utilizados para treinar os classificadores separando em base de treino e teste utilizando a técnica citada anteriormente.

Além disso, foi realizado o balanceamento da base de dados utilizando a técnica *SMOTE* (*Synthetic Minority Oversampling Technique*), que consiste em gerar novos exemplos sintéticos para a classe minoritária por meio de um pseudocódigo, de modo a balancear a base de dados (OLIVEIRA, 2013). A técnica foi utilizada para balancear a base, pois a mesma possui uma grande desproporção entre as classes, conforme apresentado na seção 3.5.

Posteriormente foi utilizado a *scikit-learn* para a criação dos classificadores *Random Forest*, *Logistic Regression* e *XGBoost*. A *scikit-learn* é uma biblioteca de código aberto para aprendizado de máquina em Python que fornece ferramentas para auxiliar na criação, treinamento e validação de classificadores de visão computacional (SILVA et al., 2018; RODRI-

GUES; LASTHAUS, 2021).

Os classificadores foram treinados utilizando a técnica de validação cruzada *k-fold*, conforme apresentada na seção 2.4, com $k = 10$. A base de dados foi dividida em 10 partes, sendo 8 partes para treinamento e 2 partes para teste. O processo foi repetido 10 vezes, sendo que cada parte foi utilizada uma vez como conjunto de teste. O resultado final consiste na média dos resultados obtidos em cada iteração (exceto os resultados da matriz de confusão, a ser apresentada no Capítulo 5).

Conforme dito anteriormente, a base de dados possui 5.531.451 registros. Diante a quantidade massiva de dados e recursos computacionais disponíveis, não foi possível executar o treinamento dos classificadores utilizando todos os dados. A memória RAM disponível no equipamento utilizado para treino (16 GB) não foi suficiente para suportar o treinamento, que apresentou falhas devido a seu esgotamento. Assim, tornou-se necessário realizar uma leitura limitada do arquivo contendo os dados preparados, utilizando um subconjunto de 1.000.000 de registros.

Assim sendo, os algoritmos classificadores foram então treinados utilizando a técnica de sobreamostragem *SMOTE* (seção 2.5) para cada um dos classificadores, na qual, ao final do processo, foi calculado a média dos resultados. Posteriormente o procedimento foi executado novamente, porém sem utilizar a técnica de *SMOTE*, de modo a permitir a comparação entre os resultados obtidos com e sem a técnica de balanceamento. Tais resultados estão presentes na seção 5.1.

Desse modo os classificadores foram treinados e seus resultados avaliados utilizando as métricas de *accuracy*, *sensitivity*, *specificity*, *FPR fraud*, *precision* e *F-score*, apresentadas na seção 2.3. A matriz de confusão também foi utilizada para avaliar os resultados obtidos para o 10º *k-fold* (seção 2.4), a fim de avaliar os resultados obtidos de maneira particular para uma parcela da base de dados, comparado a média obtida por todos os *folds* considerados nas demais métricas.

Conforme será demonstrado, os resultados apresentados na seção 5.1 foram obtidos a partir de uma base de dados desbalanceada e por isso foi necessário aplicar a técnica SMOTE. Entretanto, a base de dados obtida da plataforma possui 5,5 milhões de registros. Mediante tal cenário, é possível extrair da base original 500 mil registros de cada classe, de modo a formar uma base de dados com 1 milhão de registros naturalmente balanceados, ou seja, com a mesma quantidade de dados rotulados como inadimplentes e adimplentes, não sendo mais necessário utilizar técnicas de balanceamento.

Diante disso, a base balanceada foi também construída de mais duas formas. Para a primeira maneira foi realizado a leitura selecionada da base original em busca dos primeiros 500 mil registros classificados como inadimplentes e em seguida o processo foi repetido para os registros classificados como adimplentes. A segunda maneira foi realizada utilizando a técnica de *undersampling 'RandomUnderSampler'* (seção 2.6) que, ao contrário da técnica de *oversampling* (em português, sobreamostragem), como o SMOTE, consiste em reduzir aleatoriamente a quantidade de registros da classe majoritária, de modo a balancear a base de

dados (LIN et al., 2017; FENG et al., 2022; BARELLA, 2015). Para esta última maneira, foi possível construir uma base balanceada com 2.755.740 registros. O processo de classificação foi então realizado também para estas duas bases (totalizando quatro versões). Os resultados obtidos estão apresentados seção 5.2.

5 Resultados

Neste capítulo são apresentados os resultados obtidos com a aplicação dos algoritmos classificadores *Random Forest*, *Logistic Regression* e *XGBoost*, as quais foram submetidos a registros extraídos da base de dados disponibilizada na plataforma *Kaggle* aplicando a técnica de validação cruzada *k-fold* com k = 10 pré-processados através de técnicas de balanceamento.

5.1 Analisando os resultados utilizando SMOTE

Nesta seção apresenta-se os resultados obtidos para os algoritmos classificadores utilizando a técnica de balanceamento SMOTE, bem como para a base desbalanceada (sem SMOTE), para a base de dados com leitura limitada a 1 milhão de registros.

Analisando os resultados para o classificador *Random Forest*, o algoritmo obteve uma acurácia de 91,25% quando não utilizado o SMOTE e 90,80% quando utilizado. Para a precisão dos adimplentes temos 93,98% não utilizando e 95,90% utilizando a técnica de sobreamostragem. Para os inadimplentes temos 83,11% de precisão sem SMOTE e 77,91% utilizando a técnica de balanceamento. Uma sensibilidade para os adimplentes de 94,44% e 81,89% para os adimplentes e inadimplentes, respectivamente, sem SMOTE e 91,64% para os adimplentes e 88,26% para os inadimplentes com SMOTE. Para *F1-score* temos 94,21% para os adimplentes quando não se utiliza o SMOTE e 93,72% quando se utiliza, bem como 82,50% para os inadimplentes não utilizando SMOTE e 82,77% quando se utiliza a técnica de balanceamento. Todos os resultados podem ser observados na Tabela 2:

Tabela 2 – Média dos resultados para Random Forest

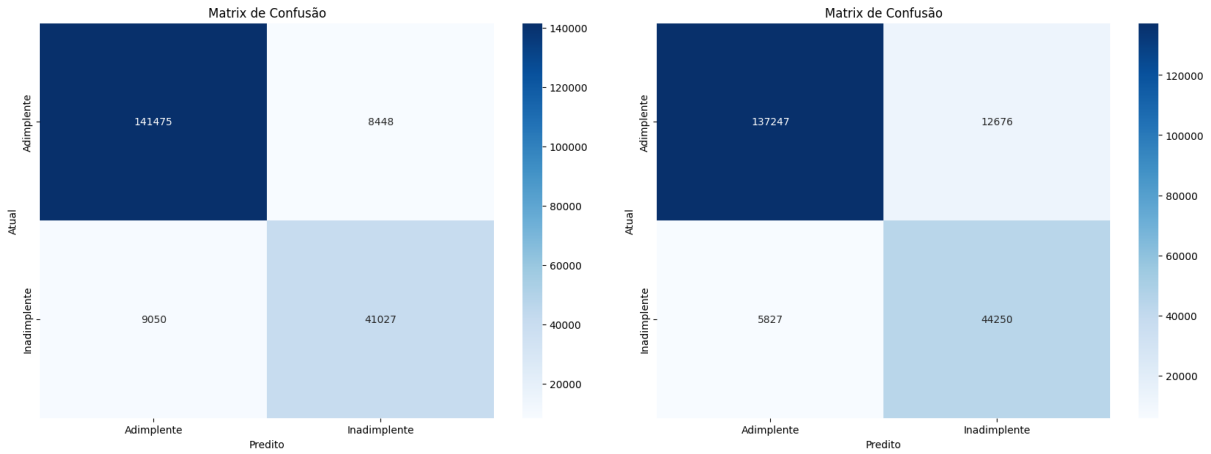
RANDOM FOREST	Sem SMOTE		Com SMOTE	
Situação	Adimplente	Inadimplente	Adimplente	Inadimplente
Sensitivity:	94,44%	81,89%	91,64%	88,26%
Specificity:	81,89%	94,44%	88,26%	91,64%
FPR fraud:	18,11%	5,56%	11,74%	8,36%
Precision:	93,98%	83,11%	95,90%	77,91%
F-score:	94,21%	82,50%	93,72%	82,77%

Fonte: elaborada pelo autor

Para a matriz de confusão, temos quatro áreas: verdadeiro negativo (*true negative* — TN), falso positivo (*false positive* — FP), falso negativo (*false negative* - FN) e verdadeiro positivo (*true positive* — TP) (MONARD; BARANAUSKAS, 2003; TOWNSEND, 1971; VISA et al., 2011). Considerando o cenário sem a utilização da técnica de sobreamostragem SMOTE, conforme citado na seção 3.5, dos 200 mil registros validados (20% da base selecionada) selecionados a partir da execução do 10º k-fold, temos que o algoritmo classificou 141.475 registros

como adimplentes quando de fato eles estão nesta situação (TN), conforme pode ser visto na Figura 44. Em contrapartida, 8.448 registros foram classificados incorretamente pois, foram considerados inadimplentes quando deveria ter sido considerado adimplente (FP). Temos também que 9.050 registros de inadimplentes foram classificados incorretamente como adimplentes (FN) e por fim, 41.027 registros foram corretamente classificados como inadimplentes (TP). Quando utilizado a técnica de SMOTE, os resultados são inferiores para classificar os adimplentes e superiores para classificar os inadimplentes, pois 137.247 registros foram classificados como verdadeiro negativo, 12.676 registros considerados falso positivo, 5.827 registros falsos negativos e 44.250 registros verdadeiros positivos, ou seja, considerados inadimplentes quando de fato estão nessa conjuntura.

Figura 44 – Matriz de confusão para Random Forest



(a) Sem SMOTE (b) Com SMOTE

Fonte: elaborada pelo autor

De modo semelhante ao apresentado anteriormente, temos as médias dos resultados para o classificador *Logistic Regression* apresentados na Tabela 3. Para este classificador foi obtido uma acurácia de 86,48% quando não utilizado a técnica de balanceamento e 85,96% quando se faz o uso do SMOTE.

Tabela 3 – Média dos resultados para Logistic Regression

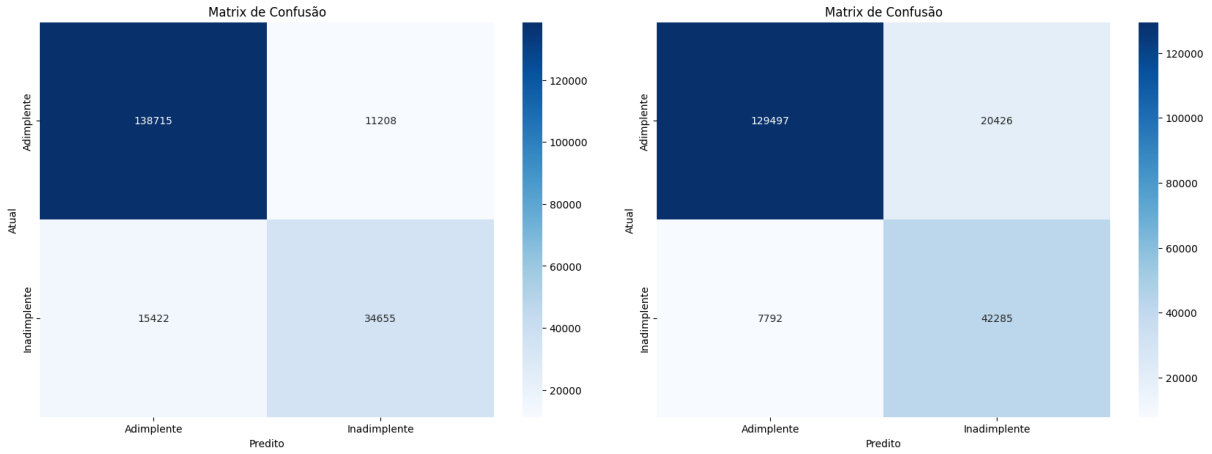
LOGISTIC REGRESSION	Sem SMOTE		Com SMOTE	
Situação	Adimplente	Inadimplente	Adimplente	Inadimplente
Sensitivity:	92,62%	68,08%	86,37%	84,73%
Specificity:	68,08%	92,62%	84,73%	86,37%
FPR fraud:	31,92%	7,38%	15,27%	13,63%
Precision:	89,68%	75,50%	94,43%	67,51%
F-score:	91,13%	71,60%	90,22%	75,14%

Fonte: elaborada pelo autor

Do mesmo modo, temos na Figura 45 a matriz de confusão para o classificador *Logistic*

Regression para o último *k-fold*:

Figura 45 – Matriz de confusão para Logistic Regression



(a) Sem SMOTE (b) Com SMOTE

Fonte: elaborada pelo autor

Por fim, da mesma maneira que os classificadores anteriores, temos a média dos resultados para o classificador *XGBoost* conforme pode ser observado na Tabela 4. Uma acurácia de 88,34% foi obtida para este classificador quando não se utiliza o SMOTE e 88,04% quando é utilizado a técnica de balanceamento.

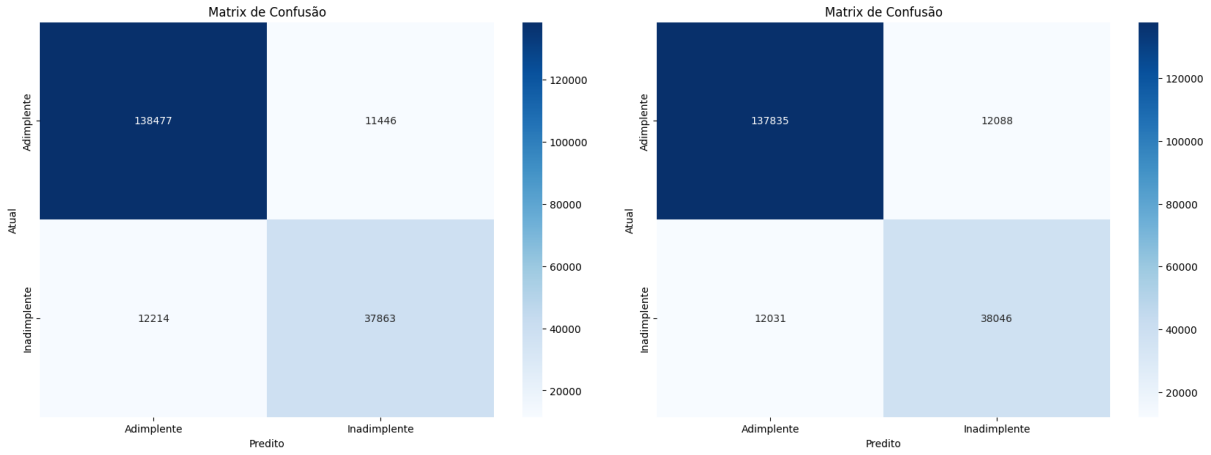
Tabela 4 – Média dos resultados para XGBoost

XGBOOST	Sem SMOTE		Com SMOTE	
Situação	Adimplente	Inadimplente	Adimplente	Inadimplente
Sensitivity:	92,44%	76,05%	92,07%	76,00%
Specificity:	76,05%	92,44%	76,00%	92,07%
FPR fraud:	23,95%	7,56%	24,01%	7,93%
Precision:	92,04%	77,07%	91,99%	76,19%
F-score:	92,24%	76,56%	92,03%	76,09%

Fonte: elaborada pelo autor

A Figura 46 apresenta a matriz de confusão contendo os resultados para o 10º *k-fold* do classificador *XGBoost*:

Figura 46 – Matriz de confusão para XGBoost



(a) Sem SMOTE (b) Com SMOTE

Fonte: elaborada pelo autor

Realizando um comparativo entre os resultados providos pelos algoritmos implementados, *Random Forest* foi o classificador que foi capaz de apresentar os melhores resultados considerando as métricas apresentadas na 2.3.

Os desempenhos dos classificadores são satisfatórios, uma vez que foram capazes de prever a inadimplência de usuários de cartão de crédito, através da utilização de algoritmos classificadores e obter acurácia de 91,25% sem SMOTE, precisão de até 95,90% para classificar os adimplentes utilizando a técnica de SMOTE e sensibilidade de 94,44% para os adimplentes não fazendo uso do SMOTE. Todos estes resultados foram obtidos para o classificador *Random Forest*, que por sua vez, obteve 77,91% de precisão para os inadimplentes para a base balanceada utilizando a técnica de SMOTE como o seu resultado menos satisfatório. Dessa forma, pode-se dizer que tal algoritmo pode representar uma importante ferramenta a ser utilizada aliada aos analistas de crédito a fim de prover melhorias na qualidade de seus trabalhos.

Analisando os resultados dos adimplentes presentes na base do 10º *k-fold* provenientes a partir da utilização da técnica de SMOTE, percebe-se que o balanceamento proporcionou uma redução na qualidade dos algoritmos classificadores, visto que nota-se uma redução de casos de adimplentes que de fato não estão inadimplentes (verdadeiro negativo), ao mesmo tempo que foi responsável por elevar a quantidade de adimplentes que foram incorretamente classificados como inadimplentes (falso positivo). Para os resultados dos clientes inadimplentes, a técnica de balanceamento proporcionou um comportamento contrário, ou seja, a quantidade de verdadeiros positivos aumentou e de falsos negativos reduziu. Para as demais métricas, a sensibilidade, *FPR fraud* e *F-score* foram moderadamente aprimoradas ao balancear a base com essa técnica. Uma possibilidade para tal comportamento pode ser explicado pelo fato de que a técnica de SMOTE realizou o aumento da quantidade de dados para os inadimplentes, uma vez que a quantidade de dados para os inadimplentes (250.387 registros) era quase 3 vezes menor (299,38%) que a quantidade de dados para os adimplentes (749.613

registros), sendo então necessário criar novos registros sintéticos para os inadimplentes.

5.2 Comparando os resultados com outras bases balanceadas

Nesta seção, os resultados obtidos a partir de uma base balanceada construída das duas maneiras conforme citado na seção 4.4 serão apresentados e comparados com os resultados da seção 5.1, como pode ser observado na Tabela 5 e Tabela 7. Em conformidade com o informado anteriormente, os classificadores *Random Forest* (RF), *Logistic Regression* (LR) e *XGBoost* (XGB) foram utilizados para treinar a base desbalanceada e balanceada utilizando a técnica de SMOTE e além disso foram também utilizados para classificar às base de dados construída à partir da leitura selecionada e fazendo uso técnica de *undersampling*, mais precisamente, *RandomUnderSampler*, conforme apresentando na seção 2.6.

Os resultados obtidos a partir da matriz de confusão estão separados de modo que os verdadeiro negativo e falso positivo estão comparados com os resultados dos clientes adimplentes e os verdadeiros positivos e falsos negativos são referentes aos clientes inadimplentes. Os resultados para a base de dados construída a partir da utilização da técnica de *undersampling*, devido ao fato da quantidade de registros ser 275,57% maior, a fim de facilitar a comparação, os resultados apresentados estão com uma proporção equivalente a mesma quantidade de registros das demais bases (1.000.000 de registros).

Para a base de dados construída a partir da leitura selecionada dos primeiros 500 mil registros de cada classe, o classificador *Random Forest* obteve 89,92%, *Logistic Regression* obteve 86,29% e *XGBoost* obteve 87,41% para a média de acurácia. Já para a base de dados construída com *undersampling*, os mesmos classificadores apresentaram respectivamente; 88,30%; 86,37% e 87,09% para a acurácia média.

Tabela 5 – Comparativo dos resultados obtidos para os adimplentes

ADIMPLENTE	Sensitivity	Specificity	FPR fraud	Precision	F-score	TN	FP
RF sem SMOTE	94,44%	81,89%	18,11%	93,98%	94,21%	141475	8448
RF com leitura selecionada	86,16%	93,68%	6,32%	93,17%	89,53%	85944	14056
RF com SMOTE	91,64%	88,26%	11,74%	95,90%	93,72%	137247	12676
XGB sem SMOTE	92,44%	76,05%	23,95%	92,04%	92,24%	138477	11446
LR sem SMOTE	92,62%	68,08%	31,92%	89,68%	91,13%	138715	11208
XGB com leitura selecionada	84,75%	90,08%	9,92%	89,52%	87,07%	84764	15236
LR com SMOTE	86,37%	84,73%	15,27%	94,43%	90,22%	129497	20426
RF com undersampling	83,86%	92,74%	7,26%	92,03%	87,76%	83844	16156
XGB com SMOTE	92,07%	75,99%	24,01%	91,99%	92,03%	137835	12088
XGB com undersampling	83,96%	90,21%	9,79%	89,56%	86,67%	84055	15945
LR com undersampling	84,39%	88,35%	11,65%	87,87%	86,09%	84399	15601
LR com leitura selecionada	84,26%	88,32%	11,69%	87,82%	86,00%	84186	15814

Fonte: elaborada pelo autor

Analisando os resultados dos usuários de cartão de crédito adimplentes para cada uma

das métricas utilizadas, todos os melhores resultados de todas para cada uma das métricas utilizadas foram obtidos para o classificador *Random Forest*. A melhor especificidade (93,68%) foi obtida ao analisar os dados da base construída a partir da leitura selecionada que também foi a responsável por obter a menor *False Positive Rate*(FPR) com 6,32% e consiste na menor proporção de casos negativos que são classificados erroneamente como positivos (ZHENG et al., 2018).

Ao classificar os dados da base de dados desbalanceada (sem SMOTE), o classificador *Random Forest* obteve uma sensibilidade de 94,44%, *F-score* de 94,21%, precisão de 93,98% e a menor quantidade de falsos positivos (8.448) e maior quantidade de verdadeiros negativos foi também obtida para este classificador e base (141.475) ao analisar os resultados obtidos da matriz de confusão.

Do ponto de vista da precisão dos classificadores, com 95,90% *Random Forest* foi também o melhor para a base de dados balanceada utilizando a técnica de SMOTE dentre todos os demais resultados. Por fim, analisando o *F-score* de todos os classificadores e bases, este classificador e esta base proporcionaram também o segundo melhor *F-score* com 93,72%. Concretizando portanto, o classificador a conquistar os melhores resultados para classificar usuários adimplentes para todas as métricas analisadas.

De outro modo, analisando os resultados menos satisfatórios, o classificador *Random Forest* para a base construída com o balanceamento *undersampling* e de tamanho 275,57% maior, apresenta a menor sensibilidade, com 83,86% e a segunda maior quantidade de falsos positivos (FP), com 16.156 registros classificados incorretamente para o 10º *k-fold*.

Para as demais métricas, *Logistic Regression* obteve a menor especificidade, *FPR fraud*, *F-score* e precisão. Para a base desbalanceada (sem SMOTE) obteve 68,08% de especificidade e 31,92% de *FPR fraud*. Este mesmo classificador para a base construída a partir da leitura selecionada obteve obteve 87,82% de precisão e 86,00% de *F-score*, sendo considerado portanto o classificador menos adequado para classificar os usuários de cartão de crédito como adimplentes.

Tabela 6 – Avaliação dos classificadores com base nos resultados para os usuários adimplentes

ADIMPLENTE	Sensitivity	Specificity	FPR fraud	Precision	F-score	TN	FP
RF sem SMOTE	3			1	3	3	3
RF com leitura selecionada		3	3				
RF com SMOTE				3	2		
XGB sem SMOTE	1	-1	-1		1	1	1
LR sem SMOTE	2	-3	-3			2	2
XGB com leitura selecionada				-1			
LR com SMOTE				2			-3
RF com undersampling	-3	2	2			-1	-2
XGB com SMOTE		-2	-2				
XGB com undersampling	-2	1	1		-1	-2	-1
LR com undersampling				-2	-2		
LR com leitura selecionada	-1			-3	-3	-3	

Fonte: elaborada pelo autor

A Tabela 6 realiza uma avaliação dos classificadores com base nos resultados obtidos para os usuários adimplente. Os três melhores resultados de cada métrica foram avaliados com uma pontuação de 1 a 3 pontos e os três piores resultados tiveram uma nota negativa atribuída de até -3 pontos, de modo que a maior nota seja atribuída ao melhor classificador para aquela métrica e a menor nota para o classificador com resultados insatisfatórios para métrica analisada em detrimento aos demais resultados. Conforme citado anteriormente, temos *Random Forest* como o melhor e *Logistic Regression* como o menos adequado para classificar usuários como adimplentes.

Para os inadimplentes, temos o comparativo apresentado na Tabela 7:

Tabela 7 – Comparativo dos resultados obtidos para os inadimplentes

INADIMPLENTE	Sensitivity	Specificity	FPR fraud	Precision	F-score	TP	FN
RF com leitura selecionada	93,68%	86,16%	13,84%	87,13%	90,28%	93681	6319
RF sem SMOTE	81,89%	94,44%	5,56%	83,11%	82,50%	41027	9050
XGB com leitura selecionada	90,08%	84,75%	15,25%	85,52%	87,74%	90132	9868
RF com SMOTE	88,26%	91,64%	8,36%	77,91%	82,77%	44250	5827
RF com undersampling	92,74%	83,86%	16,15%	85,17%	88,80%	92789	7211
LR com undersampling	88,35%	84,39%	15,61%	84,98%	86,63%	88480	11520
XGB com undersampling	90,21%	83,96%	16,04%	84,91%	87,48%	90155	9845
LR com leitura selecionada	88,32%	84,26%	15,74%	84,87%	86,56%	88246	11754
XGB sem SMOTE	76,05%	92,44%	7,56%	77,07%	76,56%	37863	12214
LR com SMOTE	84,73%	86,37%	13,63%	67,51%	75,14%	42285	7792
XGB com SMOTE	75,99%	92,07%	7,93%	76,19%	76,09%	38046	12031
LR sem SMOTE	68,08%	92,62%	7,38%	75,50%	71,60%	34655	15422

Fonte: elaborada pelo autor

A Tabela 8 apresenta a avaliação dos resultados obtidos para os usuários que são inadimplentes. Da mesma maneira que apresentado anteriormente, os algoritmos serão analisados e avaliados de acordo com os mesmos critérios utilizados para os adimplentes, diferenciando-se apenas que, para os inadimplentes ao invés de verdadeiros negativos e falsos positivos, serão analisados a quantidade de verdadeiros positivos (TP) e falsos negativos (FN) do 10º *k-fold*, respectivamente.

Tabela 8 – Avaliação dos classificadores com base nos resultados para os usuários inadimplentes

INADIMPLENTE	Sensitivity	Specificity	FPR fraud	Precision	F-score	TP	FN
RF com leitura selecionada	3			3	3	3	2
RF sem SMOTE		3	3				
XGB com leitura selecionada				2	1		
RF com SMOTE							3
RF com undersampling	2	-3	-3	1	2	2	1
LR com undersampling							
XGB com undersampling	1	-2	-2			1	
LR com leitura selecionada		-1	-1				
XGB sem SMOTE	-1	1	1			-2	-2
LR com SMOTE				-3	-2		
XGB com SMOTE	-2			-1	-1	-1	-1
LR sem SMOTE	-3	2	2	-2	-3	-3	-3

Fonte: elaborada pelo autor

Analisando a Tabela 7 e a Tabela 8, percebe-se que os melhores resultados para os usuários inadimplentes foram também obtidos para o classificador *Random Forest*. Para a base construída a partir da leitura selecionada o classificador foi capaz de classificar usuários como inadimplentes com uma sensibilidade de 93,68%, precisão de 87,13%, apresentar um *F-score* de 90,28%, obter a maior quantidade de verdadeiros positivos (TP) com 93.681 registros classificados corretamente no 10º *k-fold* e em comparação com os demais resultados, obter a segunda menor quantidade de falsos negativos (FN) com 6.319 registros classificados incorretamente no 10º *k-fold*.

Para a base de construída com a técnica de balanceamento SMOTE, esse classificador foi capaz de obter a menor quantidade de falsos negativos entre todos os classificadores utilizados (5.827 registros). Para a base de dados desbalanceada, o classificador obteve a melhor especificidade com 94,44% e *FPR fraud* de 5,56%.

Logistic Regression apresentou a menor sensibilidade entre todos os classificadores e bases utilizadas, com 68,08% para a base desbalanceada, que também foi responsável por prover o menor *F-score* para o classificador (71,60%), segunda menos precisão entre os demais resultados, bem como apresentar a menor quantidade de verdadeiros positivos (34.655) e maior quantidade de falsos negativos (15.422) para o 10º *k-fold*. Este classificador também apresentou a menor precisão (67,51%) para a base balanceada com SMOTE e o segundo

menor *F-score* (75,14%) para a base que foi balanceada utilizando a técnica de SMOTE. O classificador *Random Forest* para a base construída utilizando a técnica de *undersampling* obteve a menor especificidade (83,86%) e maior *FPR fraud* (16,15%) em comparação com os demais resultados.

Desse modo, infere-se portanto que é possível prever a inadimplência de usuários de cartão de crédito, através da utilização de algoritmos classificadores e, dentre todos os classificadores utilizados, *Random Forest* é o mais adequado para classificar um usuário de cartão de crédito como inadimplente, o que vai de encontro ao objetivo deste trabalho.

Realizando uma análise geral dos resultados apresentados, nota-se algumas contrariedades e semelhanças. Ao observar as especificidades obtidas para os clientes adimplentes, nota-se que os resultados são os mesmos para as sensibilidades obtidas para os clientes inadimplentes, do mesmo modo que a especificidade dos clientes inadimplentes é igual a sensibilidade dos clientes adimplentes. Esta situação indica que de fato a base de dados estava balanceada.

De outro modo, percebe-se que apesar da especificidade estar relacionada com a quantidade de verdadeiros negativos por exemplo, as melhores especificidades não foram encontradas para os classificadores que obtiveram a maior quantidade de verdadeiros negativos e menor falsos positivos. Tal fato pode ser explicado pela situação da base de treino do 10º *k-fold* utilizada para obter os resultados. Esse fato destaca a importância de se realizar a classificação de bases construídas de diversas maneiras e utilizando diferentes métricas, pois o resultado final pode ser influenciado dependendo da situação da base de treino utilizada.

6 Conclusão

Realizar a previsão de inadimplência de usuários de cartão de crédito fazendo uso de aprendizado de máquina de modo a definir o algoritmo classificador mais adequado mediante as características presentes na base de dados dos clientes foi a motivação deste trabalho. Os resultados para os usuários inadimplentes obtidos dos algoritmos classificadores *Random Forest*, *XGBoost* e *logistic Regression* submetidos a bases de dados construídas de diferentes formas (utilizando ou não técnicas de balanceamento), evidenciam a qualidade dos resultados providos do classificador *Random Forest* em detrimento dos demais classificadores.

Sobretudo se considerarmos que para cumprir o objetivo de identificar clientes inadimplentes, a quantidade de registros classificados como verdadeiros positivos e falsos negativos são de grande importância. Diante disso, conforme apresentado na seção 2.3, métricas como sensibilidade, precisão e consequentemente *F-score* assumem prioridade para a compreensão dos resultados obtidos. Neste cenário, *Random Forest* quando utilizado a base de dados construída com leitura selecionada dos primeiros 500.000 registros de cada rótulo, apresentou os melhores resultados. Com acurácia de 89,92%, este classificador para esta base, foi capaz de classificar os usuários inadimplentes com sensibilidade de 93,68%, especificidade de 86,16%, *FPR Fraud* de 13,84%, precisão de 87,13% e *F-Score* de 90,28%. Tal resultado para esta base pode ser explicado devido ao fato de não haver necessidade de realizar a criação de novos registros sintéticos da classe minoritária no SMOTE por exemplo.

Analisando os resultados obtidos a partir da utilização das técnicas de sobreamostragem (SMOTE) e subamostragem (*undersampling*) em relação a base desbalanceada (com 250.387 inadimplentes e 749.613 registros adimplentes), para os adimplentes, o SMOTE não foi capaz de melhorar nenhuma das métricas do *XGBoost*. Para *Random Forest* e *Logistic Regression*, foi capaz de melhorar a especificidade, precisão e *FPR fraud*. A técnica de *undersampling* utilizada foi capaz de melhorar apenas a especificidade e *FPR fraud* dos três classificadores, porém, para estas três métricas os resultados foram superiores também em comparação a técnica de sobreamostragem. Já para os resultados dos clientes inadimplentes, o SMOTE também não proporcionou melhorias para o *XGBoost*, mas melhorou a sensibilidade e *F-score* de *Random Forest* e *Logistic Regression*. A técnica de *undersampling* aumentou a sensibilidade, precisão e *F-score* dos três classificadores que, da maneira semelhante aos resultados dos adimplentes, são superiores aos resultados obtidos para a base desbalanceada e balanceada utilizando SMOTE.

Este trabalho foi capaz de prover uma compreensão sobre a utilização de algoritmos classificadores adequados para o objetivo deste trabalho e os resultados obtidos também possibilitam aferir a alteridade ao se fazer uso de técnicas de balanceamento de dados e oferece uma análise dos resultados obtidos sobre a perspectiva de diversas métricas. Além disso, neste trabalho estão presentes um comparativo de distintos algoritmos classificadores com base no estudo de trabalhos correlatos, estratégias para leitura e criação de bases de treino

e teste, técnicas de tratamento e balanceamento de dados, bem como outras estratégias, que podem ser utilizadas para a realização de trabalhos futuros similares, bem como podem ser úteis para instituições financeiras na tomada de decisões mais assertivas em relação à concessão de crédito aos seus clientes.

É possível ao futuro leitor, através da apresentação das características da base de dados e técnicas utilizadas, identificar problemas e soluções paliativas para os problemas detectados, bem como, constatar possíveis melhorias para o trabalho apresentado. Este trabalho, incluindo a base de dados, pode ser utilizado como apoio para a realização de outros projetos similares, uma vez que utiliza dados anônimos e públicos.

Ademais, não foi o objetivo deste trabalho analisar o tempo necessário para executar os algoritmos classificadores e o consumo de memória RAM utilizada durante o processo de classificação. Como sugestão de trabalho futuro, tais características poderiam ser analisadas a fim de identificar quais classificadores são melhores otimizados. Outra possibilidade de trabalho futuro seria realizar a comparação com outros classificadores como *Gradient Boosting* e *AdaBoost* que também são utilizados por trabalhos similares e que se destacaram, conforme estudo apresentado na seção 2.7.

A fim de prover continuidade a este trabalho, alguns pontos podem ser explorados a fim de complementar os resultados obtidos. A base de dados utilizada por este trabalho possui dados anonimizados, o que dificulta a compreensão dos resultados obtidos. Uma possível melhoria seria a utilização de uma base de dados com dados não anônimos, a fim de que se possa entender melhor os resultados obtidos utilizando uma ferramenta de interpretabilidade.

Além disso, devido as limitações computacionais conforme apresentado na seção 4.4, a base de dados utilizada por este trabalho não pode ser utilizada em sua totalidade. De modo a solucionar essa problemática vivenciada com a realização deste trabalho, poderia ser adotado uma estratégia que possibilite utilizar a técnica de *k-fold* aliada à divisão da base de dados em *chuncks* por exemplo, de modo a tornar possível realizar o treinamento utilizando todos os 5,5 milhões de registros da base de dados, em prol de uma possível compreensão mais assertiva para com os resultados obtidos.

Referências

AMIT, Y.; GEMAN, D. Shape quantization and recognition with randomized trees. *Neural computation*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals, v. 9, n. 7, p. 1545–1588, 1997. Citado na página 28.

ANDRADE, F. C. d. Utilização de técnica de aprendizado de máquina para auxiliar a tomada de decisão na concessão de crédito: uma aplicação em empréstimos p2p. *Sistemas de Informação-Florianópolis*, 2019. Citado nas páginas 17, 18, 37 e 38.

ANICETO, M. C. Estudo comparativo entre técnicas de aprendizado de máquina para estimação de risco de crédito. 2016. Disponível em: <https://repositorio.unb.br/handle/10482/20522>. Citado nas páginas 16, 55, 56 e 57.

ARAUJO, D. H. de. Predição com aprendizado de máquina de insumos locais do ppb para a indústria do complexo eletrônico de bens de consumo brasileira. 2020. Citado na página 64.

ASUNCION, A.; NEWMAN, D. J. *UCI machine learning repository*. 2007. Citado nas páginas 55, 56 e 57.

BARELLA, V. H. *Técnicas para o problema de dados desbalanceados em classificação hierárquica*. 2015. Tese (Doutorado) — Universidade de São Paulo, 2015. Citado na página 66.

BATISTA, M. R. S. *A utilização de algoritmos de aprendizado de máquina em problemas de classificação*. 2018. Tese (Doutorado) — Universidade de São Paulo, 2018. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55137/tde-25032019-141126/en.php>. Citado nas páginas 55, 56 e 57.

BHIMANI, J.; LEESER, M.; MI, N. Accelerating k-means clustering with parallel implementations and gpu computing. In: IEEE. *2015 IEEE High Performance Extreme Computing Conference (HPEC)*. [S.l.], 2015. p. 1–6. Citado na página 18.

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Disponível em: <https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf>. Citado nas páginas 28, 29, 30, 32 e 33.

BURMAN, P. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, Oxford University Press, v. 76, n. 3, p. 503–514, 1989. Citado na página 38.

CASTRO, J. S. d. et al. Estudo comparativo entre metodologias de aprendizado de máquina e híbridas aplicadas a risco de crédito. Fundação Escola de Comércio Álvares Penteado, 2019. Disponível em: <http://tede.fecap.br:8080/bitstream/123456789/818/3/Jane%20Sim%C3%B5es%20de%20Castro.pdf>. Citado nas páginas 56 e 57.

CHAWLA, N. V. et al. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, v. 16, p. 321–357, 2002. Disponível em: <https://doi.org/10.1613/jair.953>. Citado nas páginas 39, 40, 41 e 42.

CHEN, D. Y. *Análise de dados com Python e Pandas*. [S.l.]: Novatec Editora, 2018. Citado na página 62.

CHEN, M.-C.; HUANG, S.-H. Credit scoring and rejected instances reassigning through evolutionary computation techniques. *Expert Systems with Applications*, Elsevier, v. 24, n. 4, p. 433–441, 2003. Citado na página 15.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. [s.n.], 2016. p. 785–794. Disponível em: <https://arxiv.org/pdf/1603.02754.pdf>. Citado nas páginas 19, 20, 21, 22, 23, 24, 25, 26, 27 e 28.

COELHO, F. F.; AMORIM, D. P. de L.; CAMARGOS, M. A. de. Analisando métodos de machine learning e avaliação do risco de crédito. *Revista Gestão & Tecnologia*, v. 21, n. 1, p. 89–116, 2021. Disponível em: <http://revistagt.fpl.emnuvens.com.br/get/article/view/2089>. Citado na página 57.

COPPIN, B. *Inteligência artificial*. [S.l.]: Grupo Gen-LTC, 2017. Citado na página 17.

CRUZ, A. L. V. d. Análise de crédito e inteligência competitiva: competências requeridas ao analista de crédito bancário como profissional de inteligência. 2018. Citado na página 14.

CUNHA, J. P. Z. *Um estudo comparativo das técnicas de validação cruzada aplicadas a modelos mistos*. 2019. Tese (Doutorado) — Universidade de São Paulo, 2019. Disponível em: <https://www.teses.usp.br/teses/disponiveis/45/45133/tde-26082019-220647/publico/Dissertacao_JoaoPauloZanola.pdf>. Citado na página 38.

EL-MAGD, S. A. A.; ALI, S. A.; PHAM, Q. B. Spatial modeling and susceptibility zonation of landslides using random forest, naïve bayes and k-nearest neighbor in a complicated terrain. *Earth Science Informatics*, Springer, v. 14, n. 3, p. 1227–1243, 2021. Citado na página 18.

FACELI, K. et al. Inteligência artificial: uma abordagem de aprendizado de máquina. 2011. Citado na página 18.

FENG, H. et al. A combination of resampling method and machine learning for text classification on imbalanced data. In: SPRINGER. *Artificial Intelligence and Mobile Services–AIMS 2021: 10th International Conference, Held as Part of the Services Conference Federation, SCF 2021, Virtual Event, December 10–14, 2021, Proceedings*. [S.l.], 2022. p. 3–17. Citado na página 66.

FERNANDES, J. M. e. o. Deep Learning para Avaliação de Risco de Crédito Financeiro. Universidade Federal de Uberlândia, p. 56, jul. 2019. Disponível em: <https://repositorio.ufu.br/handle/123456789/26097>. Citado nas páginas 14, 16, 17, 56 e 57.

FERNÁNDEZ, A. et al. Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, v. 61, p. 863–905, 2018. Disponível em: <https://doi.org/10.1613/jair.1.11192>. Citado na página 42.

FONSECA, O. L. H.; NETO, F. D. M.; SOUZA, F. d. Modelos de análise de crédito utilizando técnicas de aprendizado de máquina. *Sociedade Brasileira de Matemática Aplicada e Computacional, São Carlos*, 2005. Citado na página 16.

FONTANA, É. *Introdução aos Algoritmos de Aprendizagem Supervisionada*. [S.l.]: Universidade Federal do Paraná - UFPR, 2020. Citado nas páginas 17, 18 e 19.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, Institute of Mathematical Statistics, v. 28, n. 2, p. 337–407, 2000. Citado na página 20.

FRIEDMAN, J. H. Stochastic gradient boosting. *Computational statistics & data analysis*, Elsevier, v. 38, n. 4, p. 367–378, 2002. Citado na página 22.

GANDRA, J. J. M. *DETECÇÃOO DE SPAM EM REDES SOCIAIS UTILIZANDO APRENDIZAGEM DE MÁQUINA*. 2020. Citado na página 17.

GUIMARÃES, I. A.; NETO, A. C. Reconhecimento de padrões: motodologias estatísticas em crédito ao consumidor. *RAE eletrônica*, SciELO Brasil, v. 1, p. 1–14, 2002. Citado na página 14.

HART, P. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, Citeseer, v. 14, n. 3, p. 515–516, 1968. Citado na página 53.

HENDRICKX, T. et al. Mining association rules in graphs based on frequent cohesive itemsets. In: SPRINGER. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. [S.l.], 2015. p. 637–648. Citado na página 17.

HUANG, Z. et al. Credit rating analysis with support vector machines and neural networks: a market comparative study. *Decision Support Systems*, v. 37, n. 4, p. 543–558, 2004. ISSN 0167-9236. Data mining for financial decision making. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167923603000861>. Citado na página 16.

IVAN, T. Two modifications of cnn. *IEEE transactions on Systems, Man and Communications, SMC*, v. 6, p. 769–772, 1976. Citado na página 49.

IZBICKI, R.; SANTOS, T. M. dos. *Aprendizado de máquina: uma abordagem estatística*. [S.l.]: Rafael Izbicki, 2020. Citado na página 17.

JIANG, D.; LIN, W.; RAGHAVAN, N. A novel framework for semiconductor manufacturing final test yield classification using machine learning techniques. *IEEE Access*, IEEE, v. 8, p. 197885–197895, 2020. Citado na página 64.

JUNIOR, A. *Nubank tem prejuízo líquido de US$ 45 mi no 1º tri; taxa de inadimplência fica em 4,2%*. 2022. Disponível em: <https://www.terra.com.br/economia/nubank-tem-prejuizo-liquido-de-us-45-mi-no-1-tri-taxa-de-inadimplencia-fica-em-42,1afaf1ce6feab8217a6d3f21aa023bceqscogbok.html>. Citado na página 14.

KAGGLE. *Kaggle: Your Machine Learning and Data Science Community*. 2019. Disponível em: <https://www.kaggle.com/>. Citado na página 58.

KAGGLE. *AMEX Competition*. 2022. Disponível em: <https://www.kaggle.com/datasets/hotsonhonet/amex-competition>. Citado na página 59.

KOTSIANTIS, S. B. et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, Amsterdam, v. 160, n. 1, p. 3–24, 2007. Citado na página 18.

LAURIKKALA, J. Improving identification of difficult small classes by balancing class distribution. In: SPRINGER. *Artificial Intelligence in Medicine: 8th Conference on Artificial Intelligence in Medicine in Europe, AIME 2001 Cascais, Portugal, July 1–4, 2001, Proceedings 8*. [S.l.], 2001. p. 63–66. Citado na página 53.

LEMAÎTRE, G.; NOGUEIRA, F.; ARIDAS, C. K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, v. 18, n. 17, p. 1–5, 2017. Disponível em: <http://jmlr.org/papers/v18/16-365.html>. Citado nas páginas 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54 e 55.

LIN, W.-C. et al. Clustering-based undersampling in class-imbalanced data. *Information Sciences*, Elsevier, v. 409, p. 17–26, 2017. Citado na página 66.

LOPES, R. S. Comparação de métodos de aprendizado de máquina para análise de risco de crédito. Serra, 2022. Disponível em: <https://repositorio.ifes.edu.br/handle/123456789/1755>. Citado nas páginas 56 e 57.

MAIA, P. H. C. et al. Mobipy–biblioteca para análise de padrões de mobilidade de usuários. Universidade Federal de Campina Grande, 2019. Citado na página 62.

MANI, I.; ZHANG, I. knn approach to unbalanced data distributions: a case study involving information extraction. In: ICML. *Proceedings of workshop on learning from imbalanced datasets*. [S.l.], 2003. v. 126, p. 1–7. Citado na página 46.

MARRA, V. N. et al. Previsão de dificuldades financeiras em empresas latino-americanas via aprendizagem de máquina. Universidade Federal de Uberlândia, 2019. Disponível em: <https://repositorio.ufu.br/handle/123456789/24750>. Citado na página 57.

MCKINNEY, W. pandas.dataframe.fillna. *pandas documentation*, 2021. Disponível em: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>. Citado na página 59.

MELLO, D. *Pagamentos com cartões de crédito crescem 42% no primeiro trimestre*. 2022. Disponível em: <https://agenciabrasil.ebc.com.br/economia/noticia/2022-05/pagamentos-com-cartoes-de-credito-crescem-42-no-primeiro-trimestre>. Citado na página 14.

MENASCE, M. *O que é inadimplência?* 2021. Disponível em: <https://blog.euemdia.com.br/o-que-e-inadimplencia/>. Citado nas páginas 14 e 15.

MENESES, A. C. F. Predição de necessidade de uti hospitalar para pacientes internados utilizando métodos de aprendizado de máquina. 2021. Citado nas páginas 37 e 38.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, Manole, v. 1, n. 1, p. 32, 2003. Citado na página 67.

NAQA, I. E.; MURPHY, M. J. What Is Machine Learning? In: NAQA, I. E.; LI, R.; MURPHY, M. J. (Ed.). *Machine Learning in Radiation Oncology: Theory and Applications*. Cham: Springer International Publishing, 2015. p. 3–11. ISBN 978-3-319-18305-3. Disponível em: <https://doi.org/10.1007/978-3-319-18305-3_1>. Citado na página 17.

NETO, L. *Impulsionado pelo cartão de crédito, endividamento atinge em março a máxima histórica - Portal CNC*. 2022. Disponível em: <https://www.portaldocomercio.org.br/noticias/impulsionado-pelo-cartao-de-credito-endividamento-atinge-em-marco-a-maxima-historica/420334>. Citado na página 14.

OLIVEIRA, C. S. Reamostragem e imputação de dados em caso de eventos raros. 2013. Citado na página 64.

PAIXÃO, W. S. da; TEIXEIRA, S. S. Impacto financeiro causado por inadimplência: Um estudo em uma empresa de crédito. *Revista Innovare-ISSN 2175-8247*, v. 1, 2019. Citado na página 15.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Disponível em: <https://scikit-learn.org>. Citado nas páginas 33, 34, 35, 36, 37 e 46.

PINTO, F. W. C. Classificação de reclamações do portal consumidor. gov. br utilizando aprendizagem automática. 2021. Citado na página 64.

POWERS, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020. Citado nas páginas 37 e 38.

RASCHKA, S.; MIRJALILI, V. *Python Machine Learning (Second). Birmingham B3 2PB, UK*. [S.l.]: Packt Publishing Ltd. Retrieved from https://www. packtpub. com, 2017. Citado na página 64.

RIBEIRO, R. F.; LARA, R. O endividamento da classe trabalhadora no brasil e o capitalismo manipulatório. *Serviço Social & Sociedade*, SciELO Brasil, p. 340–359, 2016. Citado na página 15.

RODRIGUES, P. H. B.; LASTHAUS, A. Técnicas de visão computacional para classificação de peças. *Revista Mackenzie de Engenharia e Computação*, v. 21, n. 1, p. 144–167, 2021. Citado nas páginas 64 e 65.

SANCHES, M. K. *Aprendizado de máquina semi-supervisionado: proposta de um algoritmo para rotular exemplos a partir de poucos exemplos rotulados*. 2003. Tese (Doutorado) — Universidade de São Paulo, 2003. Citado na página 17.

SANTOS, J. O. d.; FAMÁ, R. Avaliação da aplicabilidade de um modelo de credit scoring com variáveis sistêmicas e não-sistêmicas em carteiras de crédito bancário rotativo de pessoas físicas. *Revista Contabilidade & Finanças*, SciELO Brasil, v. 18, p. 105–117, 2007. Citado na página 15.

SAS. *Machine learning: o que é e qual sua importância?* 2022. Disponível em: <https://www.sas.com/pt_br/insights/analytics/machine-learning.html>. Citado na página 15.

SCHNEIDER, C. F. Machine learning aplicado na previsão de resultados de partidas de futebol: um estudo de caso para comparação de diferentes classificadores. 2018. Citado na página 18.

SILVA, G. L. d. et al. Desenvolvimento de rede neural de multicamadas para predição de parâmetros de soldagem. Universidade Federal Rural do Semi-Árido, 2018. Citado nas páginas 64 e 65.

SINGH, A.; HALGAMUGE, M. N.; LAKSHMIGANTHAN, R. Impact of different data types on classifier performance of random forest, naive bayes, and k-nearest neighbors algorithms. *International Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 8, n. 12, 2017. Citado na página 18.

SINHORIGNO, S.; VICENTE, R. Previsão de inadimplência de transações com cartão de crédito: um estudo comparativo. 2007. Disponível em: <https://repositorio.usp.br/item/001609352>. Citado nas páginas 56 e 57.

SMITH, M. R.; MARTINEZ, T.; GIRAUD-CARRIER, C. An instance level analysis of data complexity. *Machine learning*, Springer, v. 95, p. 225–256, 2014. Citado na página 55.

SOUZA, A. A. de. Luppar news-rec: Um recomendador inteligente de notícias. Universidade Estadual do Ceará, 2019. Citado na página 37.

TEIXEIRA, L. d. A. Métodos de regressão para aprendizado por reforço. 2016. Citado na página 18.

TOMEK, I. An experiment with the edited nearest-nieghbor rule. 1976. Citado na página 51.

TOWNSEND, J. T. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, Springer, v. 9, n. 1, p. 40–50, 1971. Citado na página 67.

VENTURA, M. M. O estudo de caso como modalidade de pesquisa. *Revista SoCERJ*, v. 20, n. 5, p. 383–386, 2007. Citado na página 58.

VISA, S. et al. Confusion matrix-based feature selection. *MAICS*, v. 710, n. 1, p. 120–127, 2011. Citado na página 67.

WAZLAWICK, R. S. *Metodologia de pesquisa para ciência da computação*. [S.l.]: Elsevier, 2009. v. 2. Citado na página 58.

WILSON, D. L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, IEEE, n. 3, p. 408–421, 1972. Citado na página 50.

YADAV, S.; SHUKLA, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In: IEEE. *2016 IEEE 6th International conference on advanced computing (IACC)*. [S.l.], 2016. p. 78–83. Citado na página 60.

YIN, R. K. *Estudo de Caso-: Planejamento e métodos*. [S.l.]: Bookman editora, 2015. Citado na página 58.

ZHENG, L. et al. Transaction fraud detection based on total order relation and behavior diversity. *IEEE Transactions on Computational Social Systems*, IEEE, v. 5, n. 3, p. 796–806, 2018. Citado na página 72.