Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Modelo de classificação para dados desbalanceados: método SMOTE e variantes

Andrielle Couto Nora

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Modelo de classificação para dados desbalanceados: método SMOTE e variantes

Andrielle Couto Nora
Orientador: Prof. Dr. Carlos Alberto Ribeiro Diniz

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Estatística.

São Carlos
Fevereiro de 2024

Federal University of São Carlos
Exact and Technology Sciences Center
Department of Statistics

Classification model for unbalanced data: SMOTE method and variants

Andrielle Couto Nora
Advisor: Prof. Dr. Carlos Alberto Ribeiro Diniz

Bachelors dissertation submitted to the Department of Statistics, Federal University of São Carlos - DEs-UFSCar, in partial fulfillment of the requirements for the degree of Bachelor in Statistics.

São Carlos
February 2024

Andrielle Couto Nora

Modelo de classificação para dados desbalanceados: método SMOTE e variantes

Este exemplar corresponde à redação final do trabalho de conclusão de curso devidamente corrigido e defendido por Andrielle Couto Nora e aprovado pela banca examinadora.

Aprovado em 29 de janeiro de 2024.

Banca Examinadora:

- Prof. Dr. Carlos Alberto Ribeiro Diniz
- Prof. Dr. Márcio Luis Lanfredi Viola
- Prof. Dr. Rafael Izbicki

Resumo

Frequentemente, em modelos de classificação, nos deparamos com bancos de dados que possuem classes muito desbalanceadas, como por exemplo: dados de diagnóstico de doenças raras, defeitos de fabricação, transações fraudulentas, etc. Treinar um modelo em um conjunto de dados com poucas observações de uma determinada classe resulta em um desempenho preditivo ruim do mesmo, especialmente para as observações pertencentes à classe minoritária. Neste Trabalho de Conclusão de Curso (TCC), apresentamos e comparamos diferentes variantes do método SMOTE (*Synthetic Minority Over-sampling TEchnique*) de sobreamostragem de dados desbalanceados utilizados em modelos de classificação, especificamente, a Regressão Logística, a fim de demonstrar como essas técnicas podem melhorar a capacidade de identificar e prever observações da classe minoritária em cenários realistas e desbalanceados, além de determinar qual combinação entre a técnica de amostragem e o modelo de classificação de Regressão Logística leva a um melhor desempenho.

Palavras-chave: *dados desbalanceados, modelo de classificação, Regressão Logística, sobreamostragem, SMOTE.*

Abstract

Often, in classification models, we encounter databases that have highly imbalanced classes, such as: rare disease diagnostic data, manufacturing defects, fraudulent transactions, etc. Training a model on a dataset with few observations of a particular class results in poor predictive performance, especially for observations belonging to the minority class. In this Undergraduate Thesis, we present and compare different variants of the Synthetic Minority Over-sampling TEchnique (SMOTE) method for oversampling imbalanced data used in classification models, specifically Logistic Regression, in order to demonstrate how these techniques can improve the ability to identify and predict observations from the minority class in realistic and imbalanced scenarios, as well as to determine which combination of sampling technique and Logistic Regression classification model leads to better performance.

Keywords: *classification model, imbalanced data, Logistic Regression, oversampling, SMOTE.*

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

É muito comum a presença de dados desbalanceados em problemas reais. Dados desbalanceados são determinados por situações em que pelo menos uma das classes é constituída por poucas observações, sendo denominada, por esta razão, de classe minoritária. Entre as inúmeras possibilidades da presença de tais dados, podemos citar detecção de fraudes, diagnóstico de doenças raras, defeitos de fabricação, ou situações relacionadas à ciência política e ciências sociais, como guerras, golpes, vetos presidenciais, entre outros.

Devido a essa natureza de desequilíbrio dos dados, podemos enfrentar problemas ao treinar um modelo estatístico, como por exemplo: (i) *overfitting*, em que o modelo estatístico se ajusta excessivamente aos dados de treinamento, concentrando-se, assim, fortemente na classe majoritária. Nestes casos, os modelos apresentam previsões precisas para a classe majoritária, nos dados de treinamento, mas podem acarretar em previsões imprecisas, principalmente para a classe minoritária, em novos conjuntos de dados, demonstrando, portanto, dificuldades na generalização; (ii) viés do modelo, ocasionado quando o conjunto de treinamento é dominado por uma classe e o modelo tende a se enviesar a favor dessa classe dominante, provocando previsões tendenciosas e não realistas, especialmente para a classe minoritária; (iii) desempenho assimétrico, já que modelo pode ter um bom desempenho para a classe majoritária, mas apresentar desempenho inferior para a classe minoritária, tornando-o ineficaz em cenários em que todas as classes são importantes; (iv) dificuldade em detectar eventos raros, uma vez que, se a classe minoritária representar um evento raro, a baixa representatividade dessas observações pode dificultar a detecção desses eventos pelo modelo.

Não é fácil explicar e prever eventos raros. King e Zeng (2001) acreditam que, entre as possíveis razões, está o fato dos procedimentos estatísticos mais populares, como a

Regressão Logística (LaValley (2008)), subestimarem acentuadamente a probabilidade de eventos raros ou das estratégias de amostragem de dados geralmente usadas serem grosseiramente ineficientes.

Uma estratégia bastante utilizada no passado na construção de amostras para o ajuste de modelos de Regressão Logística, quando os dados são desbalanceados, é selecionar uma amostra contendo todos os eventos presentes na base de dados original e sortear, via amostragem aleatória simples sem reposição, um número de observações da classe majoritária igual ou superior ao número de observações da classe minoritária (Diniz e Louzada (2013)).

Porém, atualmente, novos métodos foram desenvolvidos e estão sendo mais utilizados. Neste Trabalho de Conclusão de Curso (TCC) discutimos o SMOTE (*Synthetic Minority Over-sampling TEchnique*), uma técnica de amostragem de dados utilizada para aumentar a amostra da classe minoritária da variável resposta, evitando o *overfitting*. Para contornar isso, o SMOTE gera novas instâncias sintéticas, isto é, observações específicas ou particulares de determinada classe (neste caso, a classe minoritária), que possui atributos ou características definidas por essa classe, sendo que as instâncias de uma mesma classe compartilham o mesmo espaço de características, uma representação matemática que transforma dados brutos em vetores de características mais adequados para análise e modelagem por algoritmos de aprendizado de máquina, mas seus valores ou conteúdos nessas características podem ser diferentes, refletindo as particularidades ou estados individuais de cada observação.

Chawla *et al.* (2002) propuseram o algoritmo SMOTE e demonstraram sua eficácia em lidar com o desequilíbrio de classes, ilustrando o desempenho do novo método através de diferentes conjuntos de dados. Hernandez *et al.* (2013) compararam o desempenho de diferentes métodos de sobreamostragem e subamostragem, incluindo o SMOTE, em conjuntos de dados desbalanceados públicos, oferecendo evidências de que o uso desses métodos melhora significativamente a precisão da classe minoritária em relação ao conjunto de dados original. Kovács (2019) apresentou e discutiu em seu artigo uma comparação empírica detalhada de 85 técnicas de sobreamostragem da classe minoritária, incluindo o SMOTE e suas próprias variantes, analisando 104 conjuntos de dados desbalanceados, a fim de obter uma compreensão do desempenho dessas técnicas em diferentes tipos de conjuntos de dados, além do desempenho de diferentes princípios operacionais.

Neste estudo, exploramos o uso de modelos de classificação, especificamente, a Re-

gressão Logística, para lidar com desbalanceamento, focando na aplicação do método SMOTE e suas variantes. Utilizamos uma base de dados de transações de cartões de crédito que apresenta um desequilíbrio significativo entre transações legítimas e fraudulentas e uma base de renda do censo com um desequilíbrio mais moderado. Nosso objetivo é demonstrar como essas técnicas podem melhorar a capacidade de identificar e prever transações fraudulentas e indivíduos com uma renda superior a $50 mil por ano em cenários realistas e desbalanceados, além de determinar qual combinação entre a técnica de amostragem e o modelo de classificação de Regressão Logística leva a um melhor desempenho.

Este trabalho está organizado da seguinte maneira. No Capítulo 2, descrevemos a técnica SMOTE e suas variantes, sendo elas Borderline-SMOTE, SMOTE-Tomek, kmeans-SMOTE e ADASYN. No Capítulo 3, fizemos uma descrição e breve análise das bases de dados que foram utilizadas para a aplicação e comparação dos métodos. No Capítulo 4, apresentamos resultados das aplicações feitas utilizando os conjuntos de dados. O Capítulo 5 encerra esta monografia abordando as considerações finais do estudo em questão.

Capítulo 2

Synthetic Minority Over-sampling TEchnique e suas variantes

Neste Capítulo, descrevemos a técnica SMOTE e suas variantes, sendo elas Borderline-SMOTE, SMOTE-Tomek, kmeans-SMOTE e ADASYN.

2.1 *Synthetic Minority Over-sampling TEchnique* (SMOTE)

O *Synthetic Minority Over-sampling TEchnique* (SMOTE) (Chawla *et al.* (2002)) é uma técnica amplamente utilizada para lidar com conjuntos de dados desbalanceados, particularmente eficaz quando a classe minoritária (a classe com menos observações) da variável resposta está sub-representada no conjunto de dados. O SMOTE visa equilibrar a distribuição de classes gerando instâncias sintéticas da classe minoritária.

Essa técnica funciona da seguinte maneira. Na primeira etapa analisamos o conjunto de dados e identificamos a classe minoritária da variável resposta. Neste estágio, o SMOTE seleciona aleatoriamente uma observação da classe minoritária que servirá de base para a geração de instâncias sintéticas. Uma vez que uma observação da classe minoritária é selecionada, o SMOTE identifica seus k vizinhos mais próximos no espaço de características.

O valor de k é determinado pelo usuário e representa o número de vizinhos mais próximos a serem considerados, e a escolha adequada do valor de k é essencial para garantir que as novas instâncias geradas sejam realistas e representativas do espaço de características original. Algumas considerações e critérios comuns para determinar o valor de k são:

- **Tamanho do conjunto de dados:** o valor de k não deve ser muito grande em relação ao tamanho do conjunto de dados original, pois pode levar a uma geração excessiva de instâncias sintéticas, resultando em *overfitting* e perda de informações importantes. Em geral, k é escolhido como um valor menor em comparação com o tamanho total do conjunto de dados;

- **Balanceamento desejado:** o valor de k pode ser escolhido com base no grau de desbalanceamento do conjunto de dados original. Se o desbalanceamento for extremo, um valor de k maior pode ser necessário para gerar mais instâncias sintéticas da classe minoritária, porém, por outro lado, se o desbalanceamento for moderado, um valor de k menor pode ser suficiente;

- **Número de vizinhos minoritários disponíveis:** é importante considerar o número real de vizinhos minoritários disponíveis para cada observação da classe minoritária. Se o número de vizinhos minoritários é muito baixo em relação a k, pode ser difícil gerar instâncias sintéticas significativas. Nesse caso, é preferível reduzir o valor de k;

- **Características e distribuição dos dados:** um valor específico de k pode funcionar melhor para um determinado conjunto de dados, mas pode ser menos eficaz em outros;

- **Validação cruzada (*cross-validation*):** A validação cruzada é uma técnica de avaliação de modelos de aprendizado de máquina que visa melhorar a generalização do modelo e reduzir o viés de avaliação. Nela, o conjunto de dados é dividido em várias partes para treinamento e teste de forma iterativa. Em cada iteração, uma parte diferente é utilizada como conjunto de teste, enquanto as outras partes são usadas para treinar o modelo. Ao final, os resultados são combinados para fornecer uma estimativa mais confiável do desempenho do modelo em todo o conjunto de dados. Nesse contexto, pode ser útil realizar uma validação cruzada com diferentes valores de k para avaliar o desempenho do modelo em diferentes configurações, o que ajudará a identificar qual valor de k leva a resultados mais equilibrados e, geralmente, mais precisos;

- **Testes empíricos:** realizar testes empíricos com diferentes valores de k, isto é, o usuário pode executar o algoritmo SMOTE com diferentes valores de k e avaliar os

resultados com métricas de desempenho;

- **Consultar a literatura:** A literatura científica e estudos relacionados podem fornecer orientações sobre valores de k comumente utilizados para conjuntos de dados semelhantes.

Em resumo, a escolha do valor de k no SMOTE é uma tarefa empírica que envolve considerar as características do conjunto de dados, o grau de desbalanceamento, o tamanho do conjunto de dados e a disponibilidade de vizinhos minoritários. Experimentar com diferentes valores de k e avaliar o desempenho do modelo é uma abordagem recomendada para encontrar a configuração mais adequada.

Dessa forma, o SMOTE, então, cria instâncias sintéticas interpolando entre a observação selecionada e seus vizinhos mais próximos. No contexto da técnica SMOTE, a etapa de interpolação envolve a criação de instâncias sintéticas geradas a partir da combinação das características de uma observação minoritária selecionada com as características dos seus vizinhos mais próximos, de forma ponderada. Ele faz isso escolhendo aleatoriamente um ou mais dos vizinhos mais próximos e criando instâncias sintéticas ao longo do segmento de linha que os conecta, calculando a diferença entre a observação e um dos vizinhos selecionados e multiplicando essa diferença por um número aleatório δ. A Equação (2.1) ilustra, matematicamente, esse processo.

$$x_{sintetico} = x_i + \delta \cdot (\hat{x}_i - x_i), \tag{2.1}$$

em que x_i é a observação selecionada da classe minoritária, $\hat{x}_i$ é o vizinho selecionado dentre os k possíveis e δ é um número aleatório entre 0 e 1. Vale ressaltar que, se na base de dados utilizada tivermos p covariáveis, tanto x_i quanto $\hat{x}_i$ e $x_{sintetico}$ serão vetores p-dimensionais.

O valor aleatório δ atua como um peso que controla a extensão da interpolação, e é usado para determinar quanto da diferença de características entre a observação minoritária e seu vizinho selecionado deve ser incluído na instância sintética. Por exemplo, se o valor aleatório for 0, a instância sintética será uma réplica exata da observação minoritária, enquanto que, se o valor aleatório for 1, a instância sintética será uma réplica exata do vizinho selecionado. Valores entre 0 e 1 permitem vários graus de interpolação, combinando as características da observação minoritária e seu vizinho selecionado para criar uma nova instância sintética. O objetivo de utilizar aleatoriedade no processo de in-

terpolação é introduzir diversidade nas instâncias sintéticas geradas, reduzindo o risco de *overfitting* e aumentando a capacidade de generalização do conjunto de dados resultante.

Ressaltamos que o método do SMOTE e a Equação (2.1) são utilizados para o caso em que as covariáveis são todas contínuas, uma vez que a técnica SMOTE possui essa abordagem que envolve o cálculo de distâncias entre as observações, o que exige a utilização de atributos numéricos contínuos. Na situação de uma ou mais covariáveis ser categórica, outras variantes do SMOTE podem ser utilizadas, como o SMOTENC, usado quando há covariáveis categóricas e contínuas, ou o SMOTEN, utilizado quando todas as covariáveis são categóricas.

Dessa forma, as etapas anteriores são repetidas até que o nível desejado de sobreamostragem da classe minoritária seja alcançado. O usuário pode especificar a proporção desejada de observação de classes minoritárias para majoritárias no conjunto de dados final. Os passos do método descrito podem ser ilustrados por meio da Figura 2.1.

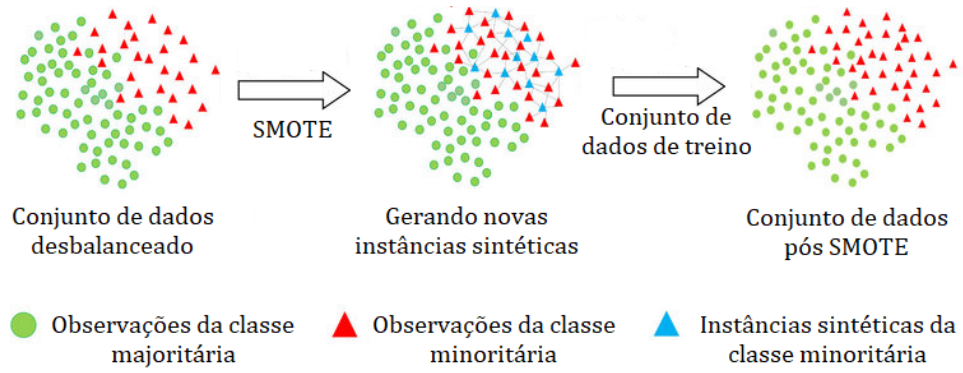

Figura 2.1: Ilustração do método SMOTE.

Fonte: Adaptado de Machine learning prediction of susceptibility to visceral fat associated diseases - Scientific Figure on ResearchGate.

As instâncias sintéticas são adicionadas ao conjunto de dados original, aumentando efetivamente o número de observações da classe minoritária. Assim, o conjunto de dados fica rebalanceado, ou seja, com uma representação mais balanceada das classes minoritária e majoritária. Depois que o conjunto de dados estiver equilibrado, podemos treinar um modelo usando o conjunto de dados aumentado e as instâncias sintéticas geradas pelo SMOTE ajudam o modelo a aprender os padrões e características da classe minoritária de forma mais eficaz.

A metodologia SMOTE possui inúmeras variantes. Neste TCC abordamos, além do próprio SMOTE, outras quatro variantes: Borderline-SMOTE, SMOTE-Tomek, kmeans-SMOTE e ADASYN.

2.2 Borderline-SMOTE

Borderline-SMOTE (Han *et al.* (2005)) é uma variante da técnica SMOTE que se concentra na geração de instâncias sintéticas da variável resposta em regiões mais próximas do limite de decisão entre as classes minoritária e majoritária, chamadas de regiões limítrofes (em inglês, *borderline*). Ele visa abordar a questão dos erros de classificação incorreta que podem ocorrer ao usar o SMOTE. A principal ideia por trás do Borderline-SMOTE é gerar seletivamente instâncias sintéticas apenas para as observações próximas às regiões limítrofes.

Existem dois tipos de observações limítrofes, as minoritárias e as majoritárias. Observações limítrofes minoritárias são observações da classe minoritária que são classificadas erroneamente como sendo da classe majoritária, ou que possuem um grande número de vizinhos da classe majoritária. Observações limítrofes majoritárias são observações da classe majoritária que são erroneamente classificadas como sendo da classe minoritária, ou que possuem um grande número de vizinhos da classe minoritária.

A primeira etapa do Borderline-SMOTE envolve a identificação das observações limítrofes dentro da classe minoritária da variável resposta que estão classificadas incorretamente ou estão próximas de serem classificadas incorretamente por meio de um classificador treinado no conjunto de dados desbalanceado original. Uma vez identificadas as observação limítrofes, instâncias sintéticas são geradas seletivamente apenas para essas observações. Para cada observação limítrofe, o SMOTE é aplicado selecionando seus k vizinhos mais próximos da mesma classe, sendo k um parâmetro definido pelo usuário e usado para determinar o número de vizinhos a serem considerados para cada observação, podendo ser escolhido segundo os mesmos critérios já descritos na técnica tradicional do SMOTE. As instâncias sintéticas são geradas como no SMOTE, interpolando entre a observação limítrofe e um de seus vizinhos mais próximos, e são adicionadas ao conjunto de dados original.

Desse modo, um esquema do Borderline-SMOTE é apresentado na Figura 2.2.

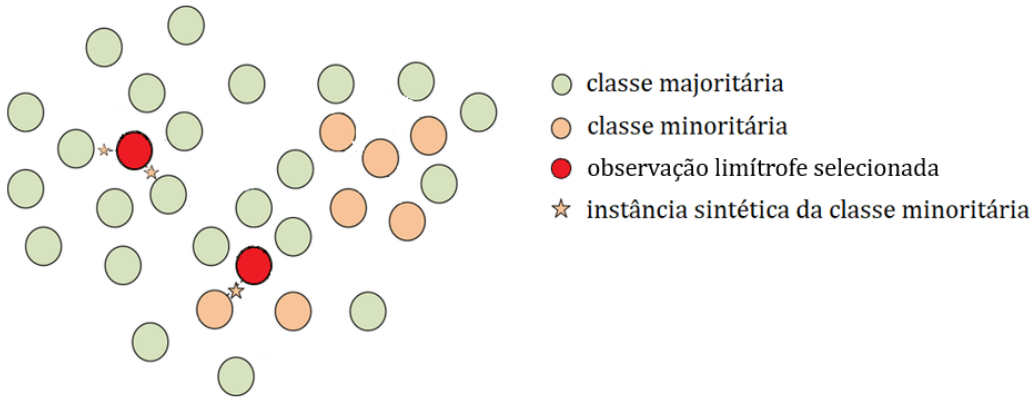

Figura 2.2: Diagrama do princípio do algoritmo Borderline-SMOTE.

Fonte: Adaptado de The real-time state identification of the electricity-heat system based on Borderline-SMOTE and XGBoost - Scientific Figure on ResearchGate.

Portanto, o SMOTE e o Borderline-SMOTE possuem o modo de seleção de observações para geração de instâncias sintéticas como a diferença mais relevante entre eles. O SMOTE gera instâncias sintéticas para todas as observações da classe minoritária, enquanto o Borderline-SMOTE se concentra apenas nas observações limítrofes. Ao fazer isso, o Borderline-SMOTE enfatiza a geração de instâncias sintéticas em regiões mais próximas do limite de decisão, o que pode ajudar a melhorar a generalização do modelo e reduzir os erros de classificação, além de que essa abordagem mais direcionada, por consequência, faz com que o Borderline-SMOTE, geralmente, seja preferido em relação ao SMOTE ao lidar com conjuntos de dados altamente desbalanceados.

2.3 SMOTE-Tomek

SMOTE-Tomek (Batista *et al.* (2003)) é uma combinação de duas técnicas: SMOTE e Tomek Links (Tomek (1976)). Ele visa abordar simultaneamente a questão do desbalanceamento e enfrentar o problema da sobreposição de observações de diferentes classes. Essa sobreposição refere-se à situação em que as observações de diferentes classes, em um conjunto de dados, estão próximas ou se sobrepõem no espaço de características. Em outras palavras, ocorre uma interseção ou sobreposição das regiões ocupadas por diferentes classes. Isso pode ser problemático, pois a presença de sobreposição dificulta a tarefa de separar corretamente as observações de classes diferentes, levando a uma maior probabilidade de erros de classificação.

Sendo assim, o SMOTE-Tomek combina os pontos fortes de ambas as técnicas para melhorar o desempenho da predição da classe minoritária. Para contextualizar, primeiramente, detalhamos um pouco o método Tomek Links.

2.3.1 Tomek Links

Tomek Links (Tomek (1976)) é um método utilizado para resolver o desbalanceamento de classe e melhorar o limite de decisão entre as classes. Ele se concentra na identificação e remoção de "*links* Tomek", os quais são pares de observações de diferentes classes (uma observação da classe minoritária e outra da classe majoritária) que são os vizinhos mais próximos uns dos outros. Essas observações são fontes potenciais de erros de classificação e, removendo esses *links*, o método visa aumentar a separação entre as classes minoritária e majoritária.

A primeira etapa do Tomek Links envolve selecionar os k vizinhos mais próximos para cada observação no conjunto de dados. Uma vez que os vizinhos mais próximos são determinados, o método procura por observações de diferentes classes que formam *links* Tomek. O Tomek Links identifica a presença de um *link* Tomek verificando se a distância entre as observações que formam o par é mínima entre todos os possíveis pares de observações de diferentes classes. Depois de identificar os *links* Tomek, o próximo estágio envolve removê-los do conjunto de dados, excluindo as observações que os compõem. A remoção dos *links* Tomek pode criar uma margem maior entre as classes, melhorando potencialmente o desempenho de classificação.

É importante destacarmos que existem diferentes abordagens em relação à remoção das observações, que podem variar de acordo com a implementação específica do método ou com as considerações do pesquisador em relação ao impacto na separação e representação das classes. Uma das abordagens comuns é remover ambas as observações que formam um *link* Tomek. Nessa abordagem, tanto as observações da classe majoritária quanto as observações da classe minoritária que compõem o *link* Tomek são removidas. Essa estratégia visa eliminar completamente a influência das observações sobrepostas na separação e na classificação das classes.

No entanto, outras variantes do método podem adotar uma abordagem que remove apenas as observações da classe majoritária que formam um *link* Tomek, por exemplo, mantendo intactas as observações da classe minoritária. O objetivo seria preservar as informações contidas nas observações da classe minoritária, enquanto ainda trata o problema

da sobreposição de observações de diferentes classes. Cada abordagem tem suas próprias vantagens e desvantagens, e é importante avaliar qual se adequa melhor ao problema em questão e aos objetivos do estudo.

Além disso, é fundamental entendermos que o Tomek Links é uma técnica de limpeza de dados e que não envolve a geração de instâncias sintéticas como no SMOTE. Ao remover seletivamente as observações que formam os *links* Tomek, o método se concentra em melhorar a separação entre as classes, em vez de equilibrar a distribuição de classes. O Tomek Links pode ser usado independentemente como uma etapa de pré-processamento antes de aplicar algoritmos de classificação ou em combinação com outras técnicas para lidar com o desbalanceamento de classe, como no caso do SMOTE-Tomek.

A Figura 2.3 representa uma ilustração da técnica em questão.

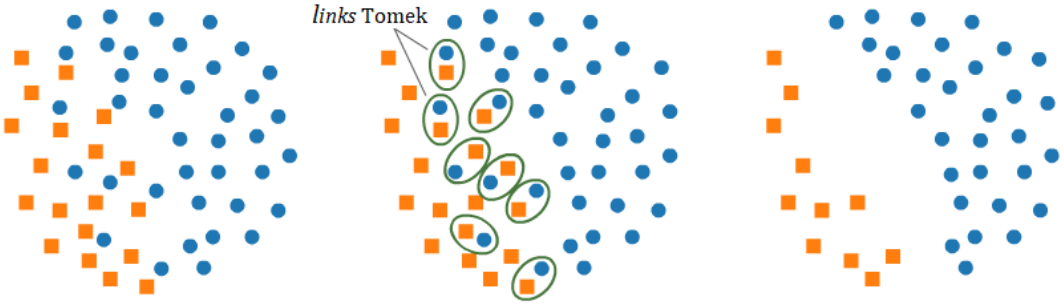

Figura 2.3: Ilustração da técnica de Tomek Links.

Fonte: Adaptado de SMOTE and Tomek Links for imbalanced data.

2.3.2 SMOTE-Tomek

Definido e detalhado o método Tomek Links, a metodologia SMOTE-Tomek funciona da seguinte maneira. O primeiro passo do SMOTE-Tomek envolve a aplicação da técnica SMOTE ao conjunto de dados desbalanceado. Instâncias sintéticas são geradas para as observações da classe minoritária, conforme explicado anteriormente na metodologia SMOTE. Esta etapa visa aumentar o número de observações da classe minoritária e equilibrar a distribuição de classes.

Depois de gerar as instâncias sintéticas, a técnica de Tomek Links é aplicada ao conjunto de dados aumentado. Nesta etapa, os *links* Tomek entre as observações das classes minoritária e majoritária são identificados e removidos do conjunto de dados. O conjunto de dados resultante apresenta uma distribuição mais equilibrada das classes minoritária e majoritária, assim como uma separação mais clara entre as mesmas, reduzindo ainda

mais as observações sobrepostas. As instâncias sintéticas geradas no estágio SMOTE contribuem para aumentar a representação da classe minoritária, enquanto a remoção dos *links* Tomek ajuda a melhorar a separação de classes.

A principal diferença entre SMOTE e SMOTE-Tomek está na etapa adicional de remoção de *links* Tomek no SMOTE-Tomek. Essa etapa ajuda a resolver o problema de observações sobrepostas e a reduzir possíveis erros de classificação. Ao combinar o SMOTE e Tomek Links, o SMOTE-Tomek visa melhorar o desempenho da predição da classe minoritária, criando uma distribuição mais equilibrada e distinta entre as classes.

2.4 Kmeans-SMOTE

O algoritmo kmeans-SMOTE (Last *et al.* (2017)) é uma variante da técnica SMOTE que incorpora o conceito de agrupamento para gerar instâncias sintéticas. Essa variante visa abordar a limitação do SMOTE em termos de seleção aleatória de vizinhos para geração de instâncias sintéticas, o que pode levar a amostras ruidosas. Uma amostra ruidosa, nesse contexto, refere-se a uma instância ou observação que contém erros, valores discrepantes ou informações irrelevantes que não representam corretamente o padrão geral do conjunto de dados. O kmeans-SMOTE usa o algoritmo de agrupamento *k-means* para selecionar vizinhos representativos e gerar instâncias sintéticas com base em suas características. Para contextualizar, primeiramente, detalhamos um pouco o método *k-means*.

2.4.1 *k-means*

O agrupamento *k-means* é um algoritmo popular utilizado para agrupar observações em grupos distintos, tendo como objetivo particionar os dados em k *clusters* (agrupamentos), em que cada observação pertence a apenas um grupo. O algoritmo tenta fazer com que as observações dentro de cada *cluster* sejam o mais semelhantes possíveis, ao mesmo tempo em que mantém os *clusters* o mais diferentes possíveis.

O primeiro passo no agrupamento *k-means* é inicializar o algoritmo. Isso envolve a seleção do número de *clusters* (k) nos quais desejamos que os dados sejam divididos. Além disso, os centroides iniciais do *cluster* são atribuídos aleatoriamente ou inicializados usando outras técnicas. No contexto do *k-means*, os centroides dos *clusters* representam os pontos centrais de cada *cluster*, calculados como a posição média de todas as observações

pertencentes a um *cluster* específico, e servem como pontos de referência que definem o centro de cada *cluster*. Durante o estágio de inicialização do algoritmo *k-means*, os centroides iniciais do *cluster* são atribuídos. Porém, existem diferentes abordagens e métodos para determinar os centroides iniciais, como selecionar aleatoriamente k observações do conjunto de dados e defini-las como centroides iniciais, ou utilizar o método de inicialização *k-means++*, o qual seleciona os centroides iniciais de forma a promover uma boa dispersão do *cluster*.

Em seguida, cada observação é atribuída ao centróide mais próximo com base na distância entre cada observação a cada centróide, normalmente a distância euclidiana. Considerando dois pontos $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ em um plano bidimensional, a distância euclidiana é dada pela Equação (2.2):

$$D(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{2.2}$$

Dessa forma, a distância entre cada observação e cada centróide é calculada, e a observação é atribuída ao *cluster* correspondente ao centróide mais próximo. Depois que todas as observações são atribuídas aos *clusters*, os centroides dos *clusters* são atualizados e os novos centroides são calculados tomando a média das observações de dentro de cada *cluster*. Esta etapa garante que o centróide represente o centro das observações em seu *cluster*. As etapas de atribuição e atualização são repetidas iterativamente até a convergência, a qual ocorre quando os centroides não mudam mais significativamente entre as iterações, ou quando um número predefinido de iterações é atingido. Uma vez que o algoritmo converge, o resultado final é um conjunto de k *clusters*, com cada observação do banco de dados tendo sido atribuída a um desses *clusters*.

É importante observar que o *k-means* é sensível às atribuições iniciais do centróide e pode convergir para diferentes soluções dependendo da sua inicialização. Para atenuar isso, o algoritmo geralmente é executado várias vezes com inicializações diferentes, e a solução com a menor soma geral de distâncias quadradas (conhecida como "soma de quadrados dentro do *cluster*" ou "inércia") é selecionada como o resultado final do agrupamento.

Além disso, determinar o valor apropriado de k (o número de *clusters*) pode ser uma tarefa desafiadora. Existem várias técnicas disponíveis, como o método do cotovelo ou a análise de silhueta, os quais ajudam a selecionar um valor ideal de k com base nas

características do conjunto de dados, mas que não são abordadas aqui.

2.4.2 *Kmeans-SMOTE*

Definido e detalhado o método *k-means*, a metodologia kmeans-SMOTE funciona da seguinte maneira. A primeira etapa do kmeans-SMOTE envolve a aplicação do algoritmo de agrupamento *k-means* às observações da classe minoritária. O algoritmo *k-means* agrupa observações semelhantes em *clusters* (agrupamentos) com base em sua similaridade de características. O número de *clusters*, indicado como k, é um parâmetro definido pelo usuário. Para cada observação da classe minoritária, o algoritmo *k-means* a atribui a um *cluster* específico.

A razão de desequilíbrio (*imbalance ratio*, IR) é um parâmetro que determina o equilíbrio entre as classes minoritária e majoritária. Por exemplo, uma razão de desequilíbrio de 0,5 significa que a classe minoritária será superamostrada até atingir metade do tamanho da classe majoritária. Essa proporção ajuda a controlar o grau de superamostragem e garante um nível desejado de equilíbrio de classe no conjunto de dados resultante. Apenas *clusters* que tenham uma alta proporção (IR $>$ 50% ou definido pelo usuário) de observações da classe minoritária são selecionados para terem mais instâncias sintéticas geradas.

Em seguida, ao invés de selecionar vizinhos aleatoriamente como no SMOTE, o kmeans-SMOTE seleciona os vizinhos mais próximos apenas do mesmo *cluster* da observação que está sendo aumentada. Isso garante que instâncias sintéticas sejam geradas a partir de observações mais próximas no espaço de características, e com características mais semelhantes. Uma vez que os vizinhos são selecionados do mesmo *cluster*, instâncias sintéticas são geradas usando a mesma técnica de interpolação do SMOTE, isto é, selecionando, aleatoriamente, um dos k vizinhos mais próximos e criando uma instância sintética em um ponto escolhido aleatoriamente ao longo do segmento de reta que conecta a observação minoritária e o vizinho selecionado.

Por fim, as instâncias sintéticas geradas são adicionadas ao conjunto de dados original, semelhantemente à técnica SMOTE, e esse conjunto de dados aumentado, agora, tem uma representação mais equilibrada das classes minoritárias e majoritárias.

Uma ilustração do método kmeans-SMOTE é mostrada na Figura 2.4, tomando nesse exemplo $k = 3$.

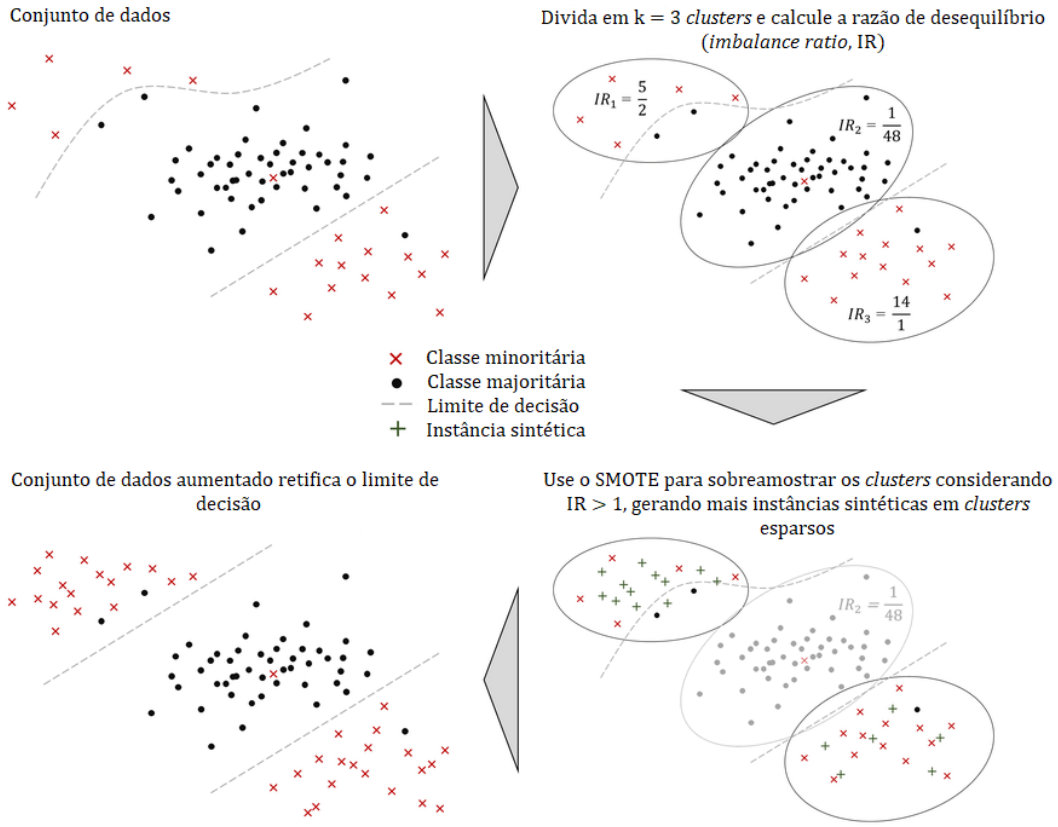

Figura 2.4: Ilustração do método kmeans-SMOTE.

Fonte: Adaptado de Oversampling for Imbalanced Learning Based on K-Means and SMOTE - Scientific Figure on ResearchGate.

Desse modo, enquanto o SMOTE seleciona, aleatoriamente, vizinhos de toda a classe minoritária, o kmeans-SMOTE aplica o agrupamento *k-means* para agrupar observações semelhantes e escolhe vizinhos apenas do mesmo *cluster*. Isso ajuda a gerar instâncias sintéticas mais representativas e consistentes com as características das observações dentro de cada *cluster*. O uso do agrupamento *k-means* no kmeans-SMOTE pode levar a melhores resultados, garantindo que instâncias sintéticas sejam geradas a partir de observações que exibem padrões e características semelhantes, e pode ajudar a reduzir o ruído introduzido pela seleção aleatória de vizinhos no SMOTE e criar instâncias sintéticas mais alinhadas com a distribuição de dados subjacente.

2.5 *Adaptive Synthetic Sampling* (ADASYN)

Adaptive Synthetic Sampling (ADASYN) (He *et al.* (2008)) é outra variante da técnica SMOTE, projetada para lidar com conjuntos de dados desbalanceados. O ADASYN aborda uma das limitações do SMOTE ajustando a geração de instâncias sintéticas com

base na distribuição de densidade das observações da classe minoritária. Ele se concentra em gerar mais instâncias sintéticas para observações mais difíceis de aprender, ou seja, observações da classe minoritária que são consideradas mais complexas ou desafiadoras para um modelo de aprendizado de máquina aprender corretamente, uma vez que, quando um conjunto de dados é desbalanceado, o desempenho de muitos algoritmos de aprendizado de máquina pode ser prejudicado, pois o modelo pode favorecer a classe majoritária devido a sua predominância. Essas observações são identificadas com base na distribuição de densidade das observações da classe minoritária.

Em regiões em que a densidade é menor, ou seja, onde há menos observações da classe minoritária, o ADASYN prioriza a criação de mais instâncias sintéticas para equilibrar a representação das classes. Sendo assim, o fato da técnica concentrar seus esforços em reforçar as áreas mais desafiadoras para o modelo, permitindo que ele aprenda melhor a fronteira de decisão e evite o viés em direção à classe majoritária, faz com que o ADASYN se adapte à distribuição de dados de maneira mais eficaz e, assim, proporcione resultados mais realistas para a classe minoritária.

Consideremos, primeiramente, um conjunto de dados com m observações, sendo m_s e m_l o número de observações das classes minoritária e majoritária, respectivamente, em que $m_s \leq m_l$ e $m_s + m_l = m$. A proporção de observações da classe minoritária em relação à majoritária é dada por meio da Equação (2.3):

$$d = \frac{m_s}{m_l}, \tag{2.3}$$

em que $d \in (0, 1]$.

Se $d < d_{th}$, sendo que d_{th} é um valor limite predefinido para o máximo grau tolerado da relação de desequilíbrio de classe, inicializamos o algoritmo. Assim, calculamos o número total de instâncias sintéticas da classe minoritária a serem geradas por meio da Equação (2.4):

$$G = (m_l - m_s) \cdot \beta, \tag{2.4}$$

sendo $\beta \in [0, 1]$ um parâmetro utilizado para especificar o nível de equilíbrio desejado após a geração de instâncias sintéticas. O valor $\beta = 1$ significa que um conjunto de dados totalmente balanceado é criado após o processo de generalização.

Para cada observação x_i pertencente à classe minoritária, encontramos seus k vizinhos mais próximos e calculamos o valor da razão r_i, chamada taxa de densidade, definida pela Equação (2.5):

$$r_i = \frac{\Delta_i}{k}, \quad i = 1, \ldots, m_s, \tag{2.5}$$

em que Δ_i é o número de observações entre os k vizinhos mais próximos que pertencem à classe majoritária, e $r_i \in [0, 1]$. A taxa de densidade r_i mede o nível de desbalanceamento entre as classes minoritária e majoritária, comparando a densidade das observações vizinhas da classe minoritária com a densidade das observações vizinhas da classe majoritária.

Em seguida, normalizamos r_i de acordo com $\hat{r}_i = \frac{r_i}{\sum_{i=1}^{m_s} r_i}$, para que $\hat{r}_i$ represente uma probabilidade $\left(\sum_i \hat{r}_i = 1 \right)$. Por meio da Equação (2.6), calculamos, então, o número de instâncias sintéticas que precisam ser geradas para cada observação da classe minoritária x_i:

$$g_i = \hat{r}_i \cdot G, \tag{2.6}$$

sendo G o número total de instâncias sintéticas da classe minoritária a serem geradas, como definido na Equação (2.4).

Para cada observação x_i da classe minoritária, geramos g_i instâncias sintéticas escolhendo, aleatoriamente, uma observação da classe minoritária, $\hat{x}_i$, dentre os k vizinhos mais próximos da observação x_i, e gerando a instância sintética por meio da Equação (2.7):

$$x_{sintetico} = x_i + \delta \cdot (\hat{x}_i - x_i), \tag{2.7}$$

em que $x_{sintetico}$ é a nova instância sintética, x_i e $\hat{x}_i$ são duas observações da classe minoritária pertencentes à mesma vizinhança e δ é um número aleatório ($\delta \in [0, 1]$). Vale ressaltar que, se na base de dados utilizada tivermos p covariáveis, tanto x_i quanto $\hat{x}_i$ e $x_{sintetico}$ serão vetores de dimensão p.

Uma ilustração do método ADASYN pode ser vista nas Figuras 2.5 e 2.6.

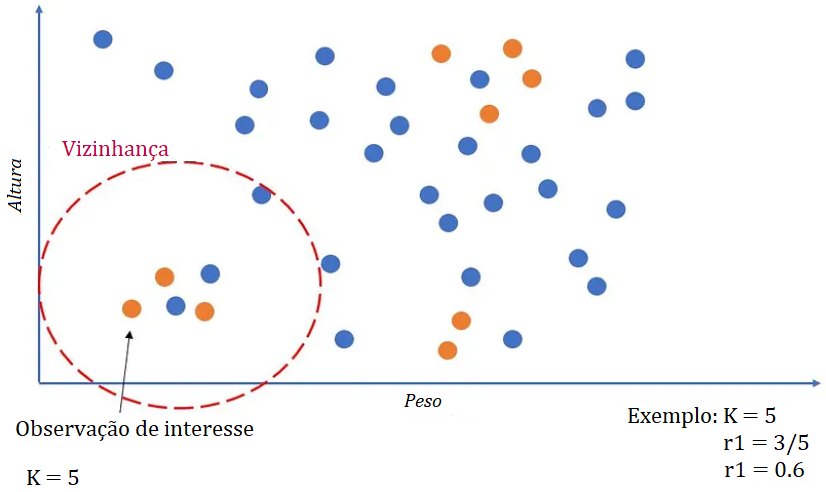

Figura 2.5: Ilustração do método ADASYN na etapa de encontrar os k vizinhos mais próximos da observação de interesse pertencente à classe minoritária e do cálculo da taxa de densidade (r_i).

Fonte: Adaptado de Fixing Imbalanced Datasets: An Introduction to ADASYN.

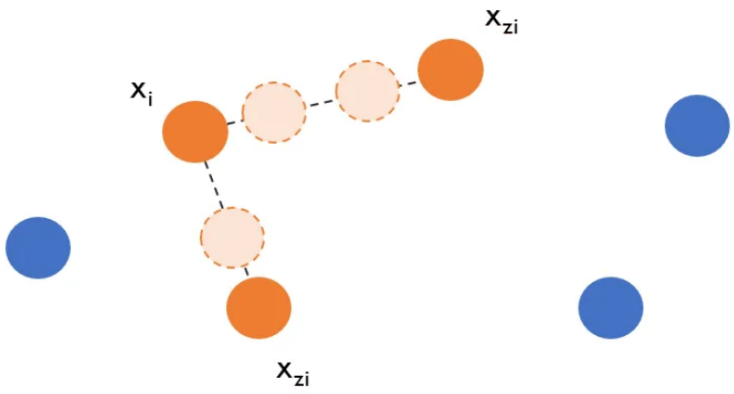

Figura 2.6: Ilustração do método ADASYN na etapa de gerar g_i instâncias sintéticas escolhendo aleatoriamente uma observação da classe minoritária, x_{zi}, dentre os k vizinhos mais próximos da observação x_i, e fazer uma combinação linear de x_i e x_{zi}.

Fonte: Adaptado de Fixing Imbalanced Datasets: An Introduction to ADASYN.

Os métodos SMOTE e ADASYN se diferenciam, principalmente, pelo estágio de geração de instâncias sintéticas, uma vez que, enquanto o SMOTE gera um número igual de instâncias sintéticas para as observações da classe minoritária selecionadas, o ADASYN adapta a geração com base na taxa de densidade. Observações com maior taxa de densidade, indicando maior dificuldade de aprendizado, recebem um número maior de instâncias sintéticas. Essa natureza adaptativa do ADASYN permite que a técnica se concentre mais nas observações desafiadoras, melhorando a capacidade do modelo de aprender com a classe minoritária. O ADASYN é, particularmente, útil ao lidar com

conjuntos de dados com vários graus de desbalanceamento de classes e observações sobrepostas, já que, ao ajustar a geração de instâncias sintéticas com base na distribuição da densidade local, o ADASYN pode efetivamente lidar com conjuntos de dados com características complexas.

2.6 Comparação das variantes

De forma geral, a escolha do método depende das características específicas do conjunto de dados e do problema de desbalanceamento em questão. Embora esses métodos compartilhem o objetivo de lidar com o desequilíbrio de classe, eles diferem em suas técnicas e abordagens específicas. Para facilitar a discussão e a comparação, vamos abordar os métodos com base em seus princípios operacionais mais característicos.

Kovács (2019) comparou algumas variantes do SMOTE por meio de categorias criadas com base nas características mais marcantes de cada método. Definimos as categorias em questão a seguir.

1-) Amostragem ordinária: os métodos desta categoria implementam a mesma amostragem que o SMOTE, com base na suposição de que os pontos ao longo dos segmentos de reta que conectam as observações minoritárias vizinhas pertencem à classe minoritária. Dessa forma, novas instâncias sintéticas são geradas pela Equação (2.8).

$$x_{sintetico} = x_i + \delta \cdot (\hat{x}_i - x_i), \tag{2.8}$$

em que x_i é a observação selecionada da classe minoritária, $\hat{x}_i$ o vizinho selecionado dentre os k possíveis e δ um número entre 0 e 1. Vale ressaltar que, se na base de dados utilizada tivermos p covariáveis, tanto x_i quanto $\hat{x}_i$ e $x_{sintetico}$ serão vetores p-dimensionais;

2-) Borderline: os métodos desta categoria implementam alguns passos explícitos para identificar observações minoritárias próximas ao limite de decisão (regiões limítrofes) e gerar novas instâncias sintéticas nas vizinhanças dessas observações limítrofes;

3-) Uso de uma densidade de amostragem: os métodos desta categoria atribuem algumas pontuações de importância às observações minoritárias e a normalização dessas pontuações resulta na densidade de amostragem $d_i, i = 1, \ldots, m_s$, usada

no processo de amostragem, em que m_s refere-se ao número total de observações na classe minoritária. Particularmente, d_i indica a proporção de novas instâncias sintéticas a serem geradas na vizinhança da i-ésima observação minoritária. Além disso, apenas as técnicas que usam algumas pontuações de importância avançadas estão incluídas neste categoria. Por exemplo, diversas técnicas usam o número de observações majoritárias nas vizinhanças de observações minoritárias como pontuação de importância: um grande número de vizinhos majoritários indica que muitas observações minoritárias adicionais precisam ser geradas na vizinhança para facilitar a classificação correta da observação minoritária;

4-) Uso de agrupamento (*clustering*): os métodos desta categoria usam técnicas de agrupamento para agrupar observações semelhantes e, em seguida, fazer a sobreamostragem de forma independente para cada *cluster*;

5-) Remoção de ruído: os métodos dessa categoria focam em lidar com amostras ruidosas, as quais, nesse contexto, referem-se a instâncias ou observações que contêm erros, valores discrepantes ou informações irrelevantes que não representam corretamente o padrão geral do conjunto de dados;

6-) Mudança da classe majoritária: os métodos dessa categoria alteram no número de observações da classe majoritária.

Com base nas categorias definidas anteriormente, podemos, resumidamente, comparar os métodos por meio da Tabela 2.1.

Tabela 2.1: Comparação dos métodos de sobreamostragem e as atribuições de categoria caracterizando seus principais princípios operacionais.

Método	Amostragem ordinária	Borderline	Uso de uma densidade de amostragem	Uso de agrupamento (clustering)	Remoção de ruído	Mudança da classe majoritária
SMOTE	X					
Borderline-SMOTE	X	X				
SMOTE-Tomek	X				X	X
kmeans-SMOTE				X		
ADASYN	X		X			

Em resumo, todos esses métodos abordam o problema de desequilíbrio de classe gerando instâncias sintéticas para a classe minoritária. O Borderline-SMOTE concentra-se na geração de instâncias sintéticas perto do limite de decisão, eficaz quando o limite de decisão entre as classes é bem definido. O mesmo gera instâncias sintéticas que são mais informativas e úteis para melhorar a classificação da classe minoritária, porém que podem

introduzir algum grau de ruído, uma vez que são geradas próximas ao limite de decisão, podendo ser classificadas erroneamente.

Já o SMOTE-Tomek combina sobreamostragem (por meio do método SMOTE) com subamostragem (por meio do método Tomek Links), tendo como objetivo melhorar a separação de classes enquanto remove observações potencialmente ruidosas ou enganosas, melhorando, assim, a qualidade dos dados de treinamento. Entretanto, O SMOTE-Tomek pode descartar algumas observações da classe minoritária que não estão envolvidas com os *links* Tomek, mas que ainda são relevantes para a classificação.

Por sua vez, kmeans-SMOTE incorpora o agrupamento *k-means* para seleção da vizinhança, ou seja, para identificar *clusters* dentro da classe minoritária e gerar instâncias sintéticas dentro de cada *cluster*. Ele leva em consideração as vizinhanças locais, potencialmente gerando instâncias sintéticas que melhor representam a distribuição de dados subjacente, e pode ser eficaz quando a classe minoritária possui *clusters* ou subgrupos distintos. Contudo, o kmeans-SMOTE baseia-se na suposição de que o agrupamento *k-means* pode identificar com precisão os *clusters* significativos dentro da classe minoritária e, portanto, depende da precisão desse algoritmo de agrupamento, além de que pode introduzir complexidade adicional e sobrecarga computacional devido à etapa de agrupamento em si.

Por fim, o ADASYN ajusta adaptativamente a distribuição de instâncias sintéticas com base na taxa de densidade de observações da classe minoritária, gerando mais instâncias sintéticas para observações da classe minoritária que são mais difíceis de aprender, ou seja, aquelas em áreas com menos observações da classe minoritária. Este método é adequado para conjuntos de dados desbalanceados com vários graus de desequilíbrio em diferentes regiões, visto que adapta-se ao desequilíbrio local e gera mais observações para instâncias minoritárias desafiadoras. Porém, como o ADASYN gera instâncias sintéticas com base em regiões de baixa densidade, ele pode ser mais sensível a amostras ruidosas ou *outliers* (valores atípicos ou extremos que se desviam significativamente do padrão geral de um conjunto de dados) presentes nessas regiões, o que pode levar a criação de instâncias sintéticas que não são representativas da classe minoritária, resultando em uma piora do desempenho do modelo. Além disso, em alguns casos, o ADASYN pode gerar muitas instâncias sintéticas para a classe minoritária, o que pode levar a um problema de sobreamostragem excessiva, podendo diluir a importância das observações originais e das informações genuínas da classe minoritária, afetando negativamente a capacidade do

modelo de generalizar para novos dados.

Dessa forma, a eficácia desses métodos pode variar dependendo do conjunto de dados e das características do desbalanceamento. Geralmente, é recomendado experimentar diferentes técnicas e avaliar seu desempenho para determinar a abordagem mais adequada para um problema específico.

Capítulo 3

Bases de Dados

Neste Capítulo, apresentamos uma descrição das bases de dados utilizadas para a aplicação e comparação dos métodos abordados no Capítulo 2.

3.1 Base de Dados de Fraude

A detecção de fraude em transações de cartões de crédito é uma área crucial para a segurança financeira, e os bancos e as instituições financeiras buscam identificar atividades fraudulentas para proteger seus clientes. A crescente complexidade das táticas de fraude exige abordagens inovadoras para identificar comportamentos suspeitos. Um desafio comum é lidar com conjuntos de dados altamente desbalanceados, nos quais as transações legítimas superam significativamente as fraudulentas. Isso pode afetar negativamente o desempenho dos modelos de classificação tradicionais.

Nesse contexto, o banco de dados utilizado foi obtido por meio da plataforma Kaggle (2023) e contém transações feitas com cartões de crédito em setembro de 2013 por titulares de cartões europeus e apresenta transações que ocorreram em dois dias. Além disso, possui 492 fraudes em um total de 284.807 transações, ou seja, é um conjunto de dados altamente desbalanceado, no qual a classe positiva (fraudes) representa 0,1727% de todas as transações.

A base contém apenas variáveis numéricas de entrada que são o resultado de uma transformação PCA, que refere-se à aplicação da Análise de Componentes Principais (em inglês, *Principal Component Analysis* ou PCA), uma técnica de redução de dimensionalidade amplamente usada em análise de dados e aprendizado de máquina para transformar um conjunto de variáveis correlacionadas em um novo conjunto de variáveis não correla-

cionadas. Esses componentes são ordenados de forma que o primeiro componente tenha a maior variabilidade possível, o segundo componente seja o segundo maior em variância e assim por diante. O PCA busca capturar a maior parte da variabilidade dos dados usando menos variáveis, o que ajuda a simplificar a análise e reduzir o risco de *overfitting* em modelos de aprendizado de máquina. No entanto, devido a problemas de confidencialidade, não foram fornecidas as características originais e mais informações de contexto sobre os dados. As características V1, V2, ..., V28 são os componentes principais obtidos com a transformação PCA, e as únicas características que não foram transformadas com PCA são "*Time*" (Tempo) e "*Amount*" (Quantia). A característica "*Time*" contém os segundos decorridos entre uma transação qualquer e a primeira transação no conjunto de dados, porém não será usada nesse trabalho. A característica "*Amount*" é o valor da transação. A característica "*Class*" (Classe) é a variável resposta e assume o valor 1 em caso de fraude e 0 caso contrário.

3.1.1 Análise Descritiva

Para um melhor conhecimento das variáveis e dos dados do estudo, realizamos uma análise descritiva por meio do *software* Python. Primeiramente, começamos pela variável resposta "*Class*". A Figura 3.1 representa a porcentagem e a frequência de transações fraudulentas e não fraudulentas.

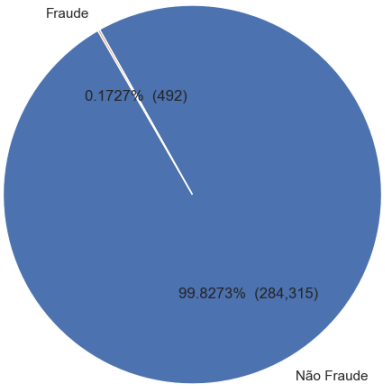

Figura 3.1: Porcentagem e frequência de transações fraudulentas e não fraudulentas.

Fonte: Elaborado pelo autor.

Notamos que 99,8273% da base (284.315 observações) é composta por transações não fraudulentas, enquanto que apenas 0,1727% da base (492 observações) é composta por transações fraudulentas. Sendo assim, temos um conjunto de dados extremamente desba-

lanceado e que justifica o uso de métodos para corrigir esse problema.

Em seguida, analisamos a covariável "*Amount*" (valor da transação) usando um histograma normalizado com a função de densidade estimada por kernel sobreposta (Figura 3.2), sendo que a estimação da função de densidade por kernel (KDE, do inglês *Kernel Density Estimate*) é uma técnica estatística usada para estimar a função de densidade de probabilidade subjacente a um conjunto de dados. Além disso, calculamos algumas medidas descritivas (Tabela 3.1).

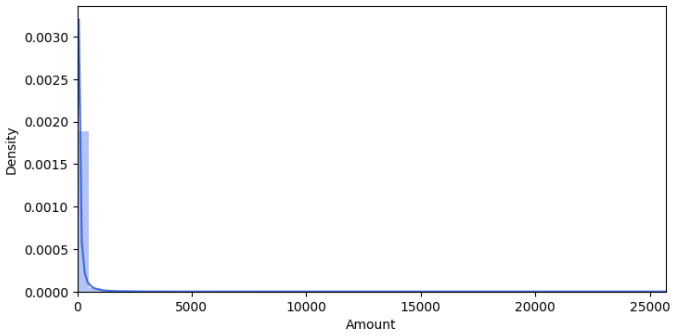

Figura 3.2: Gráfico da densidade da covariável "*Amount*".

Fonte: Elaborado pelo autor.

Tabela 3.1: Tabela de medidas descritivas da covariável "*Amount*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
88,3496	250,1201	0,0000	5,6000	22,0000	77,1650	25.691,1600

Como podemos observar pela Figura 3.2 e pelos resultados na Tabela 3.1, a covariável "*Amount*" possui uma forte assimetria à direita, com muitos valores discrepantes fazendo com que a distribuição tenha uma cauda pesada na direção de valores maiores, e os valores das transações variam entre 0 e 25.691,1600, com uma média de 88,3496, concentrando seus valores entre 5,6000 e 77,1650.

Sobre as demais covariáveis, como mencionado anteriormente, são variáveis numéricas de entrada resultantes de uma transformação PCA e, devido a problemas de confidencialidade, não foram fornecidas as características originais e maiores informações de contexto sobre os dados. Também é importante ressaltar que, para implementar uma transformação PCA, as características precisam ser previamente padronizadas devido à sensibilidade em relação à escala das variáveis, uma prática comum para garantir que a PCA funcione corretamente. Sendo assim, assumimos que as características V1, V2, ..., V28 foram padronizadas antes de aplicar a transformação PCA.

3.2 Base de Dados de Renda do Censo Americano

A análise da renda com base nos dados do censo desempenha um papel essencial na compreensão e avaliação socioeconômica das populações, e tem implicações significativas em áreas como planejamento econômico, políticas públicas e distribuição de recursos. Ao abordar esta questão, enfrentamos o desafio de determinar padrões e fatores que influenciam a renda, utilizando métodos inovadores para a análise de dados. Essa abordagem é crucial para compreender as dinâmicas socioeconômicas e informar decisões estratégicas em diversos setores.

Nesse contexto, o banco de dados utilizado foi obtido por Becker e Kohavi (1996), extraído da base de dados do Censo americano de 1994, com o objetivo de predizer se a renda de um indivíduo ultrapassa $50 mil por ano. Os registros foram selecionados com critérios específicos, garantindo dados razoavelmente limpos. Além disso, possui 7.841 indivíduos com mais de $50 mil de renda por ano em um total de 32.561 indivíduos, ou seja, 24,08% do conjunto de dados, o qual possui um desbalanceamento.

A base contém variáveis numéricas e categóricas, mas apenas as variáveis numéricas foram utilizadas nesse estudo. Foram elas: "*age*", "*fnlwgt*", "*capital_gain*", "*capital_loss*", "*hours_per_week*" e "*income*". A variável "*age*" (idade) se refere à idade dos indivíduos; "*fnlwgt*" (peso final) é o peso final atribuído aos indivíduos na amostra, utilizado para estimar a população total; "*capital_gain*" (ganho de capital) representa os ganhos financeiros provenientes de investimentos ou transações de capital; "*capital_loss*" (perda de capital) indica as perdas financeiras provenientes de investimentos ou transações de capital; "*hours_per_week*" (horas por semana) refere-se ao número de horas que os indivíduos trabalham por semana; e "*income*" (renda) indica se a renda do indivíduo ultrapassa $50 mil por ano (ou outra unidade monetária específica).

3.2.1 Análise Descritiva

Para um melhor conhecimento das variáveis e dos dados do estudo, realizamos uma análise descritiva por meio do *software* Python. Primeiramente, começamos pela variável resposta "*income*". A Figura 3.3 apresenta a porcentagem e a frequência de rendas superiores e inferiores a $50 mil por ano.

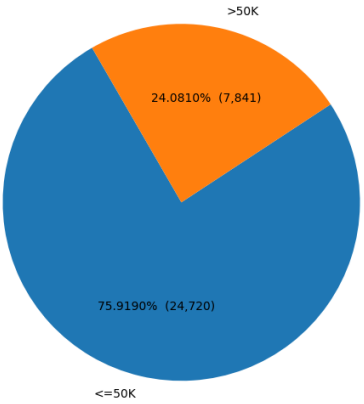

Figura 3.3: Porcentagem e frequência de rendas superiores e inferiores a $50 mil por ano.

Fonte: Elaborado pelo autor.

Notamos que 75,9190% da base (24.720 observações) é composta por rendas inferiores a $50 mil por ano, enquanto que apenas 24,0810% da base (7.841 observações) é composta por rendas superiores a $50 mil por ano. Sendo assim, temos um conjunto de dados desbalanceado e que justifica o uso de métodos para corrigir esse problema.

Em seguida, analisamos a covariável "*age*" (idade) usando um histograma normalizado com a função de densidade estimada por kernel sobreposta (Figura 3.4). Além disso, calculamos algumas medidas descritivas (Tabela 3.2).

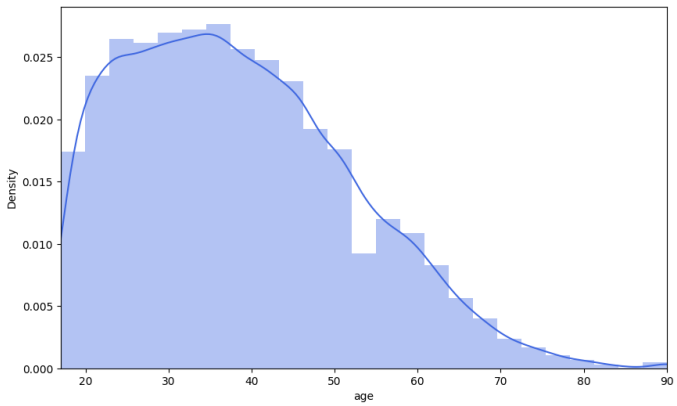

Figura 3.4: Gráfico da densidade da covariável "*age*".

Fonte: Elaborado pelo autor.

Tabela 3.2: Tabela de medidas descritivas da covariável "*age*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
38,5816	13,6404	17	28	37	48	90

Como podemos observar pela Figura 3.4 e pelos resultados na Tabela 3.2, a covariável "*age*" possui uma assimetria à direita, com muitos valores discrepantes fazendo com que

a distribuição tenha uma cauda pesada na direção de valores maiores, e os valores das idades variam entre 17 e 90 anos, com uma média de 38,5816, concentrando seus valores entre 28 e 48 anos.

Analisamos o comportamento da covariável "*fnlwgt*" (peso final) usando um histograma normalizado com a função de densidade estimada por kernel sobreposta (Figura 3.5) e uma Tabela com algumas medidas descritivas (Tabela 3.3).

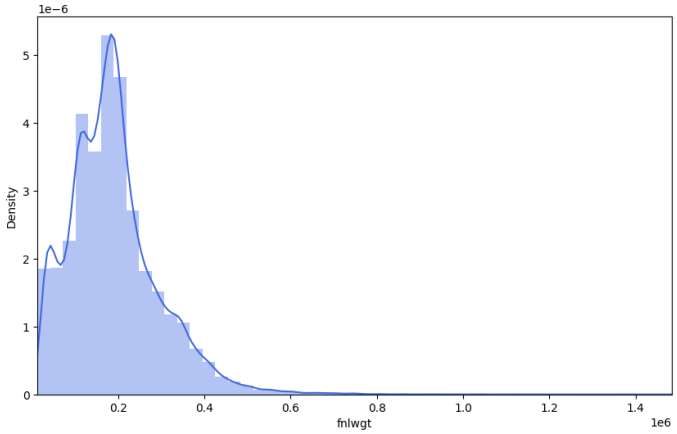

Figura 3.5: Gráfico da densidade da covariável "*fnlwgt*".

Fonte: Elaborado pelo autor.

Tabela 3.3: Tabela de medidas descritivas da covariável "*fnlwgt*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
$1{,}8978 \cdot 10^5$	$1{,}0555 \cdot 10^5$	$1{,}2285 \cdot 10^4$	$1{,}1783 \cdot 10^5$	$1{,}7836 \cdot 10^5$	$2{,}3705 \cdot 10^5$	$1{,}4847 \cdot 10^6$

Como podemos observar pela Figura 3.5 e pelos resultados na Tabela 3.3, a covariável "*fnlwgt*" possui uma forte assimetria à direita, com muitos valores discrepantes fazendo com que a distribuição tenha uma cauda pesada na direção de valores maiores, e os valores variam entre $1{,}2285 \cdot 10^4$ e $1{,}4847 \cdot 10^6$, com uma média de $1{,}8978 \cdot 10^5$.

A próxima covariável observada foi a "*capital_gain*" (ganho de capital), também por meio de um histograma normalizado com a estimativa da função de densidade por kernel sobreposta (Figura 3.6) e uma Tabela com algumas medidas descritivas (Tabela 3.4).

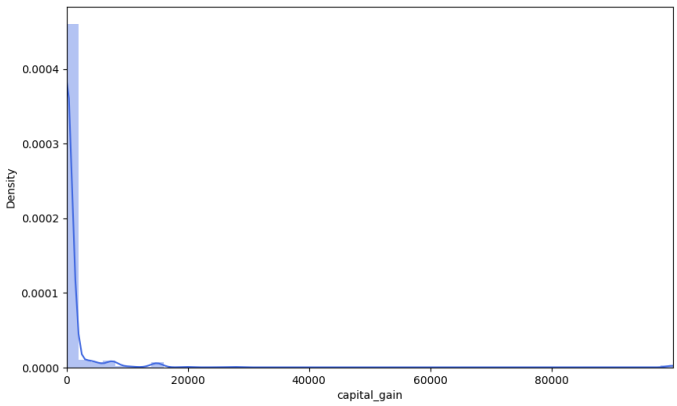

Figura 3.6: Gráfico da densidade da covariável "*capital_gain*".

Fonte: Elaborado pelo autor.

Tabela 3.4: Tabela de medidas descritivas da covariável "*capital_gain*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
1.077,6488	7.385,2921	0	0	0	0	99.999

Por meio da Figura 3.6 e pelos resultados na Tabela 3.4, percebemos que a covariável "*capital_gain*" possui uma forte assimetria à direita, com a presença de muitos valores discrepantes. Ademais, os valores de ganho financeiro variam entre 0 e 99.999, com uma média de aproximadamente 1.078 e com uma concentração de valores em 0.

Analisamos em seguida o histograma normalizado com uma estimativa da função de densidade por kernel sobreposta (Figura 3.7) e a Tabela com medidas descritivas (Tabela 3.5) da covariável "*capital_loss*" (perda de capital).

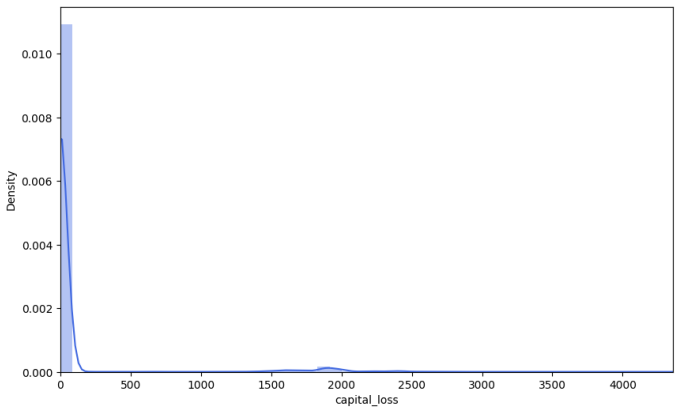

Figura 3.7: Gráfico da densidade da covariável "*capital_loss*".

Fonte: Elaborado pelo autor.

Tabela 3.5: Tabela de medidas descritivas da covariável "*capital_loss*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
87,3038	402,9602	0	0	0	0	4.356

A Figura 3.7 e os resultados na Tabela 3.5 mostraram que a covariável "*capital_loss*" possui uma forte assimetria à direita, com a presença de muitos valores discrepantes. Ademais, os valores de perda financeira variam entre 0 e 4.356, com uma média de aproximadamente 87 e com uma concentração de valores em 0.

Por fim, observamos o histograma normalizado com a estimativa da função de densidade por kernel sobreposta (Figura 3.8) e a Tabela com medidas descritivas (Tabela 3.6) da covariável "*hours_per_week*" (horas por semana).

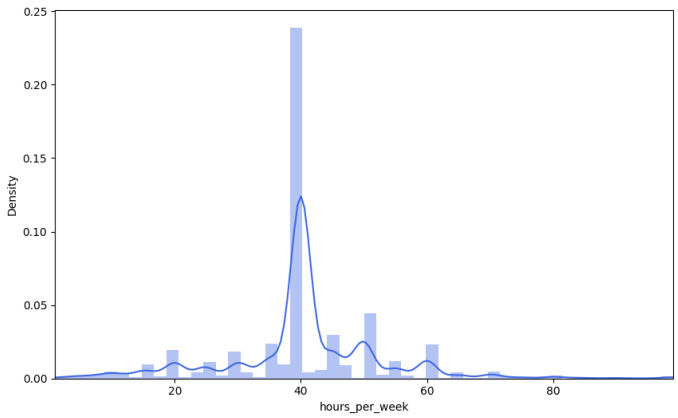

Figura 3.8: Gráfico da densidade da covariável "*hours_per_week*".

Fonte: Elaborado pelo autor.

Tabela 3.6: Tabela de medidas descritivas da covariável "*hours_per_week*".

Média	**Desvio Padrão**	**Mínimo**	**1° Quartil**	**Mediana (2° Quartil)**	**3° Quartil**	**Máximo**
40,4375	12,3474	1	40	40	45	99

Por meio da Figura 3.8 e dos resultados na Tabela 3.6, observamos que a covariável "*hours_per_week*" tem a presença de muitos valores discrepantes tanto à direta quanto à esquerda. Ademais, o número de horas trabalhadas por semana varia de 1 a 99 horas, com uma média de aproximadamente 40 horas e com uma concentração de valores entre 40 e 45 horas.

Capítulo 4

Aplicação das Técnicas de Sobreamostragem

Neste Capítulo, apresentamos os resultados da aplicação das técnicas nos conjuntos de dados de fraude de cartão de crédito e de renda do censo, descritos no Capítulo 3. Toda a análise foi elaborada por meio da utilização da linguagem de programação Python (códigos disponibilizados no Apêndice A).

Iniciamos a análise escolhendo padronizar algumas covariáveis. Na base de dados de fraude, padronizamos a covariável "*Amount*" utilizando a função `RobustScaler()`, presente na biblioteca `sklearn.preprocessing` do Python. A função dimensiona os dados usando medidas estatísticas robustas, como a mediana e os quartis, em vez da média e do desvio padrão, o que a torna mais resistente a valores extremos que poderiam distorcer o escalonamento, e é uma escolha preferível quando os dados têm muitos *outliers*. A análise descritiva presente no Capítulo 4 nos mostra que a covariável "*Amount*" tem valores com uma grande variação, possuindo uma forte assimetria à direita. Além disso, observamos a presença de muitos valores discrepantes, os quais sugeriram que o `RobustScaler()` seria adequado, uma vez que não é tão influenciado por *outliers*, tornando-o mais robusto para situações como essa. Paralelamente, na base de dados de renda do censo, padronizamos as covariáveis "*age*", "*fnlwgt*", "*capital_gain*", "*capital_loss*" e "*hours_per_week*", todas por meio também da função `RobustScaler()`, uma vez que notamos a presença de muitos valores discrepantes nessas covariáveis e de assimetria.

Após isso, fizemos a divisão dos conjuntos de dados em treinamento e teste, utilizando uma porcentagem de 70% para os conjuntos de treinamento e 30% para os conjuntos de teste. Ademais, utilizamos um parâmetro chamado "`stratify`" para garantir que

as proporções das classes presentes nos dados originais fossem mantidas nas divisões de treinamento e teste. No caso, como estamos trabalhando com conjuntos de dados de classificação que possuem uma variável resposta binária, cujo número de observações na classe positiva é muito menor do que aquele na classe negativa, fazer isso ajuda a garantir que tanto o conjunto de treinamento quanto o conjunto de teste tenham proporções semelhantes de ambas as classes.

4.1 Regressão Logística

Começamos ajustando um modelo de Regressão Logística diretamente nos dados de treinamento desbalanceados. Primeiramente, abordamos uma questão importante no ajuste de uma Regressão Logística: a escolha de um ponto de corte.

4.1.1 Ponto de Corte Ótimo

Com base em Cloud (2023), na Regressão Logística usamos as probabilidades previstas para classificar observações em duas categorias, geralmente positiva e negativa, mas, no nosso caso, em 0 (transação não fraudulenta/renda inferior a $50 mil por ano) ou 1 (transação fraudulenta/renda superior a $50 mil por ano). O ponto de corte é o valor da probabilidade usada na classificação nessas categorias. Por padrão, o ponto de corte é definido como 0,5, o que significa que qualquer observação com probabilidade predita igual ou superior a 0,5 é classificada como 1, enquanto aquelas abaixo são classificadas como 0. No entanto, o ponto de corte ótimo pode variar dependendo do problema. Em algumas situações, é fundamental encontrar o ponto de corte que otimiza o desempenho do modelo.

Para determinar esse ponto, é importante definir alguns conceitos e métricas, as quais também foram utilizadas na avaliação do desempenho do modelo. Apresentamos a Tabela 4.1, conhecida como Matriz de Confusão, a qual apresenta os resultados possíveis de uma avaliação de um modelo de classificação cuja variável resposta é binária.

Tabela 4.1: Apresentação dos resultados possíveis de uma avaliação de um modelo de classificação cuja variável resposta é binária.

		Valor Verdadeiro	
		Positivo (1)	**Negativo (0)**
Valor Previsto pelo Modelo	**Positivo (1)**	Verdadeiro Positivo (VP)	Falso Positivo (FP)
	Negativo (0)	Falso Negativo (FN)	Verdadeiro Negativo (VN)

Além disso, é importante saber que:

- **Verdadeiro Positivo (VP):** caso em que o modelo previu os valores positivos corretamente;
- **Falso Positivo (FP):** caso em que o modelo previu os valores positivos incorretamente;
- **Verdadeiro Negativo (VN):** caso em que o modelo previu os valores negativos corretamente;
- **Falso Negativo (FN):** caso em que o modelo previu os valores negativos incorretamente.

A partir dessas definições, as métricas utilizadas foram Precisão, Recall, F1-score e AUC-ROC (Área sob a Curva da Característica Operacional do Receptor), definidas como:

- **Precisão:** a métrica de Precisão, também conhecida como Valor Preditivo Positivo (VPP), é uma medida que avalia a proporção de previsões de valores positivos corretas feitas pelo modelo em relação ao total de previsões positivas. Sua fórmula é dada pela Equação (4.1).

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}}. \tag{4.1}$$

- **Recall:** a métrica Recall, também conhecido como Sensibilidade ou Taxa de Verdadeiros Positivos (TVP), mede a proporção de casos de valores previstos pelo modelo como positivos que foram identificados corretamente em relação ao número total de casos de valores verdadeiros positivos. Sua fórmula é dada pela Equação (4.2).

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}}. \tag{4.2}$$

- **F1-score:** a métrica F1-score combina a Precisão e o Recall em uma única medida, proporcionando um equilíbrio entre essas duas métricas ao fazer uma média harmônica entre as mesmas, e é calculado usando a Equação (4.3):

$$\text{F1-score} = \frac{2 \cdot \text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}} = \frac{\text{VP}}{\text{VP} + \frac{1}{2}(\text{FP}+\text{FN})}. \tag{4.3}$$

  O F1-score varia de 0 a 1, sendo que o valor de F1-score mais próximo de 1 indica um modelo com alta Precisão e alto Recall, ou seja, um equilíbrio entre a identificação de Verdadeiros Positivos (VP) e a minimização de Falsos Negativos (FN) e Falsos Positivos (FP), enquanto que o valor de F1-score mais próximo de 0 indica um modelo com baixa Precisão e/ou baixo Recall;

- **AUC-ROC (Área sob a Curva da Característica Operacional do Receptor):** a métrica AUC-ROC é amplamente usada para avaliar o desempenho de modelos de classificação binária, especialmente quando se lida com desbalanceamento de classes. A AUC-ROC mede a capacidade do modelo de distinguir entre as classes positiva e negativa em diferentes níveis de ponto de corte que separa as previsões nessas duas classes. A curva ROC é um gráfico que mostra a Taxa de Verdadeiros Positivos (Recall) em função da Taxa de Falsos Positivos cujos pontos da curva são obtidos variando-se o ponto de corte de classificação. A AUC-ROC é a área sob essa curva, que varia de 0 a 1. Quanto maior a AUC-ROC, melhor a capacidade do modelo de distinguir entre as classes, ou seja, quanto mais próximo o valor da AUC-ROC estiver de 1, melhor será o desempenho do modelo, indicando que ele tem uma boa capacidade de classificar corretamente os casos positivos e negativos, mesmo em diferentes níveis de ponto de corte.

Dessa forma, como o F1-score é, particularmente, relevante para um cenário de detecção de fraudes, em que é importante encontrar um compromisso entre a identificação de fraudes verdadeiras e a minimização de Falsos Positivos, oferecendo um balanço entre Precisão e Recall, decidimos por dar uma importância maior a ele e, assim, determinar o ponto de corte ótimo buscando aquele que maximiza o F1-score. Assim, por uma questão de uniformização, usamos esse mesmo critério para ambas as bases de dados.

A importância disso está na busca pelo equilíbrio entre Precisão e Recall. Um ponto de corte ótimo permite que adaptemos o modelo às necessidades práticas do problema, minimizando erros de classificação. Dessa forma, ao encontrarmos o ponto de corte ótimo,

conseguimos melhorar a qualidade das previsões da Regressão Logística, tornando-a mais relevante para a aplicação específica. Esse ajuste ajuda a calcular com precisão métricas de desempenho, como Precisão, Recall, F1-score e AUC-ROC, garantindo que o modelo atenda adequadamente aos requisitos do problema em questão.

4.1.2 Resultados

Ajustamos um modelo de Regressão Logística diretamente nos dados de treinamento desbalanceados e, uma vez que determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos métricas de performance do modelo.

Dessa forma, a Tabela 4.2 mostra os resultados das métricas descritas anteriormente aplicadas no conjunto de teste associado a cada conjunto de dados. Como comparação, a Tabela 4.3 apresenta os resultados das métricas caso o ponto de corte utilizado fosse o padrão, de 0,5.

Tabela 4.2: Tabela com métricas de desempenho do modelo de Regressão Logística ajustado aplicadas no conjunto de teste.

Base de Dados	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,8400	0,8544	0,5946	0,7012	0,7972
Renda do Censo	0,3000	0,5572	0,6314	0,5920	0,7361

Tabela 4.3: Tabela com métricas de desempenho do modelo de Regressão Logística ajustado aplicadas no conjunto de teste para o **ponto de corte padrão**.

Base de Dados	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,5935	0,6216	0,6073	0,8104
Renda do Censo	0,7043	0,3980	0,5086	0,6725

Os resultados mostram que o ponto de corte ótimo, o qual indica o limite de probabilidade acima do qual as observações são classificadas como positivas e abaixo do qual são consideradas negativas, parece refletir o desbalanceamento das bases, sendo determinado como 0,8400 para a base de fraudes e como 0,3000 para a base de renda do censo. Vale ressaltar que, ao escolher um ponto de corte mais elevado, favorecemos a Precisão em detrimento do Recall, o que significa que o modelo será mais criterioso ao rotular uma observação como positiva. Em comparação com deixar o ponto de corte como sendo o

padrão, de 0,5, observamos que, de modo geral, o modelo apresenta melhores desempenhos quando mudamos o ponto de corte e buscamos um que otimiza o problema. No nosso caso, quando buscamos um ponto de corte que maximize o F1-score.

Portanto, analisando os resultados com os pontos de corte ótimos (Figura 4.2), para a base de fraude, temos um indicativo de uma boa Precisão, cujo valor é 85,44%, o que significa que a maioria das previsões positivas do modelo são feitas corretamente. O Recall de 59,46% destaca a capacidade do modelo em identificar efetivamente os casos positivos reais. O F1-score, uma métrica que equilibra Precisão e Recall, atinge 70,12%, indicando um bom equilíbrio entre essas duas medidas. Além disso, o valor da área sob a curva ROC (AUC-ROC) é de 79,72%, sugerindo uma boa capacidade do modelo em separar as classes positiva e negativa. Por sua vez, para a base de renda do censo, notamos que as métricas ficaram mais baixas, indicando que o modelo possui um desempenho moderado a bom na tarefa de classificação, com valores razoáveis de Precisão, Recall, F1-score e AUC-ROC.

4.2 SMOTE

O próximo passo foi, antes de ajustar a Regressão Logística, aplicar o SMOTE no conjunto de dados de treinamento para equilibrar a distribuição de classes, gerando instâncias sintéticas da classe minoritária a fim de lidar com o fato do conjunto de dados ser extremamente desbalanceado. Para isso, a biblioteca `imblearn.over_sampling` do *software* Python foi utilizada.

Dois parâmetros da função `SMOTE()`, denominados hiperparâmetros, configurações ajustáveis que não são aprendidas diretamente pelo modelo durante o treinamento, mas que afetam o desempenho e o comportamento do modelo, são definidos antes do treinamento e desempenham um papel crucial na otimização do modelo de aprendizado de máquina e, assim, são importantes ao aplicar o SMOTE. O primeiro, chamado "`sampling_strategy`", é usado para controlar a estratégia de amostragem que o algoritmo do SMOTE irá adotar, ou seja, a proporção do tamanho da classe minoritária em relação à majoritária. Já o segundo, chamado "`k_neighbors`", é usado para controlar o número de vizinhos próximos a serem considerados ao gerar instâncias sintéticas para a classe minoritária.

Testamos a técnica SMOTE para diferentes combinações de valores dos parâmetros

“sampling_strategy” e “k_neighbors”. Dessa forma, para cada combinação, ajustamos um modelo de Regressão Logística, determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos as métricas de desempenho no conjunto de teste a fim de determinar o valor mais adequado para cada um desses dois parâmetros.

Escolhemos a melhor combinação com base na métrica F1-score e as disponibilizamos na Tabela 4.4. Em caso de empate, ordenamos com base na métrica AUC-ROC. Como já dito, essas métricas foram priorizadas pois o F1-score é, particularmente, relevante para um cenário de detecção de fraudes por oferecer um balanço entre Precisão e Recall e o AUC-ROC por indicar a capacidade de discriminar bem entre as classes.

Tabela 4.4: Tabela com a melhor combinação de “sampling_strategy” e “k_neighbors” classificada com base na métrica F1-score.

Base de Dados	“sampling_strategy”	“k_neighbors”	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,0448	10	0,9100	0,8667	0,7905	0,8269	0,8952
Renda do Censo	0,7723	9	0,5000	0,5445	0,6582	0,5960	0,7418

Com base nisso, para a base de fraude, obtemos o parâmetro “sampling_strategy” igual a 0,0448 e o parâmetro “k_neighbors” igual a 10 como estratégia. Sendo assim, o tamanho da classe minoritária é 4,48% do tamanho da majoritária, sendo que anteriormente ela era 0,1725%. Para a base de renda do censo, obtemos “sampling_strategy” igual a 0,7723 e “k_neighbors” igual a 9 e, assim, o tamanho da classe minoritária passou de 31,72% do tamanho da majoritária para 77,23%. Ademais, os pontos de corte ótimos foram determinados como 0,9100 e 0,5000, indicando o limite de probabilidade acima do qual as observações são classificadas como positivas e abaixo do qual são consideradas negativas, elevando-se para a base de fraude e permanecendo no padrão de 0,5 para a base de renda do censo. Em geral, os resultados mostram que o modelo de fraude está apresentando um desempenho razoavelmente bom, com o F1-score, Precisão e Recall elevados e servindo como indicativos de que o modelo está fazendo um equilíbrio entre a identificação de casos positivos e a minimização de falsos positivos e falsos negativos, e com a AUC-ROC elevada indicando uma boa capacidade do modelo em separar as classes positiva e negativa. O modelo de renda do censo apresenta um desempenho moderado, com as métricas em patamares inferiores a 0,75.

4.3 Borderline-SMOTE

O próximo passo foi aplicar o Borderline-SMOTE nos conjuntos de dados de treinamento para equilibrar a distribuição de classes, gerando instâncias sintéticas da classe minoritária em regiões limítrofes. Para isso, a biblioteca `imblearn.over_sampling` do *software* Python foi utilizada.

Três hiperparâmetros da função `BorderlineSMOTE()` foram importantes ao aplicar o Borderline-SMOTE. O primeiro, chamado "`sampling_strategy`", é usado para controlar a estratégia de amostragem que o algoritmo irá adotar, ou seja, a proporção do tamanho da classe minoritária em relação à majoritária. Já o segundo, chamado "`k_neighbors`", é usado para controlar o número de vizinhos próximos a serem considerados ao gerar instâncias sintéticas para a classe minoritária. Por fim, o parâmetro chamado "`m_neighbors`"é usado para controlar o número de vizinhos próximos a serem considerados para determinar se uma amostra da classe minoritária está em uma região limítrofe.

Testamos a técnica Borderline-SMOTE para diferentes combinações de valores de "`sampling_strategy`", "`k_neighbors`" e "`m_neighbors`". Dessa forma, para cada combinação, ajustamos um modelo de Regressão Logística, determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos as métricas de performance no conjunto de teste a fim de determinar os valores mais adequados para esses três parâmetros.

Sendo assim, escolhemos a melhor combinação com base na métrica F1-score e a disponibilizamos na Tabela 4.5. Em caso de empate, ordenamos com base na métrica AUC-ROC.

Tabela 4.5: Tabela com a melhor combinação de "`sampling_strategy`", "`k_neighbors`" e "`m_neighbors`" classificada com base na métrica F1-score.

Base de Dados	"sampling_strategy"	"k_neighbors"	"m_neighbors"	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,3970	9	6	0,9700	0,8657	0,7838	0,8227	0,8918
Renda do Censo	0,4182	3	5	0,3700	0,5450	0,6539	0,5945	0,7404

Para a base de fraude, obtemos o parâmetro "`sampling_strategy`" igual a 0,3970, o parâmetro "`k_neighbors`" igual a 9 e o parâmetro "`m_neighbors`" igual a 6 como estratégia. Sendo assim, o tamanho da classe minoritária é 39,70% do tamanho da majoritária, proporção maior que a anterior, com valor de 0,1725%. Para a base de renda do censo, obtemos "`sampling_strategy`" igual a 0,4182, "`k_neighbors`" igual a

3 e "m_neighbors" igual a 5, e o tamanho da classe minoritária passou de 31,72% do tamanho da majoritária para 41,82%. Os resultados mostram que o modelo está apresentando um desempenho bom para a base de fraude, uma vez que todas as métricas estão razoavelmente elevadas, e um desempenho moderado para a base de renda do censo, já que as métricas permanecem abaixo de 0,75.

4.4 SMOTE-Tomek

Aplicamos o SMOTE-Tomek nos conjuntos de dados de treinamento para equilibrar a distribuição de classes visando enfrentar o problema da sobreposição de observações de diferentes classes. Para isso, utilizamos três bibliotecas, a `imblearn.combine` para o algoritmo `SMOTETomek()`, a `imblearn.over_sampling` para o algoritmo `SMOTE()` e a `imblearn.under_sampling` para o algoritmo `TomekLinks()`, todas do *software* Python.

Três hiperparâmetros foram importantes ao aplicar o SMOTE-Tomek. Um deles é um parâmetro do próprio algoritmo `SMOTETomek()`, e os demais são parâmetros do algoritmo `SMOTE()`, lembrando que o primeiro passo do SMOTE-Tomek envolve a aplicação da técnica SMOTE ao conjunto de dados desbalanceados para, após, a técnica Tomek Links ser aplicada ao conjunto de dados aumentado. Sendo assim, o primeiro hiperparâmetro, chamado "`sampling_strategy`", é usado para controlar a estratégia de amostragem que o algoritmo `SMOTETomek()` irá adotar, ou seja, corresponde à razão desejada entre o número de observações na classe minoritária sobre o número de observações na classe majoritária após a reamostragem. Já o segundo, também denominado "`sampling_strategy`", é usado para controlar a estratégia de amostragem que o algoritmo `SMOTE()` irá adotar. Por fim, o parâmetro chamado "`k_neighbors`" é usado para controlar o número de vizinhos próximos a serem considerados ao gerar instâncias sintéticas para a classe minoritária, também no algoritmo `SMOTE()`.

Testamos a técnica SMOTE-Tomek para diferentes combinações de valores de "`sampling_strategy`" (SMOTE-Tomek),"`sampling_strategy`" (SMOTE) e "`k_neighbors`" (SMOTE). Dessa forma, para cada combinação, ajustamos um modelo de Regressão Logística, determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos as métricas de desempenho no conjunto de teste a fim de determinar os valores mais adequados para esses três parâmetros.

Assim, escolhemos a melhor combinação com base na métrica F1-score e a disponibilizamos na Tabela 4.6. Em caso de empate, ordenamos com base na métrica AUC-ROC.

Tabela 4.6: Tabela com a melhor combinação de "`sampling_strategy`" (SMOTE-Tomek), "`sampling_strategy`" (SMOTE) e "`k_neighbors`" (SMOTE) classificada com base na métrica F1-score.

Base de Dados	"sampling_strategy" (SMOTE-Tomek)	"sampling_strategy" (SMOTE)	"k_neighbors" (SMOTE)	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,0050	0,0050	1	0,6100	0,8571	0,7297	0,7883	0,8648
Renda do Censo	0,4000	0,4000	1	0,3900	0,5592	0,6301	0,5926	0,7363

Analisando a Tabela 4.6, observamos que o valor dos parâmetros "`sampling_strategy`" (SMOTE-Tomek) e "`sampling_strategy`" (SMOTE) é igual a a 0,0050 e o parâmetro "`k_neighbors`" (SMOTE) igual a 1 como estratégia para a base de fraude. Para a base de renda do censo, obtemos "`sampling_strategy`" (SMOTE-Tomek) e "`sampling_strategy`" (SMOTE) ambos iguais a 0,4000 e "`k_neighbors`" (SMOTE) igual a 1. Ademais, o limite de probabilidade acima do qual as observações são classificadas como positivas e abaixo do qual são consideradas negativas (ponto de corte ótimo), para a base de fraude, foi determinado como 0,6100, bem abaixo do que as demais técnicas. Os resultados das métricas calculadas mostram que o modelo de fraude está apresentando um desempenho bom, uma vez que todas estão razoavelmente elevadas, e que o modelo de renda do censo apresenta um desempenho moderado, já que as métricas permanecem abaixo de 0,74.

4.5 Kmeans-SMOTE

Aplicamos o kmeans-SMOTE nos conjuntos de dados de treinamento, o qual usa o algoritmo de agrupamento *k-means* para selecionar vizinhos representativos e gerar instâncias sintéticas com base em suas características. Para isso, as bibliotecas `imblearn.over_sampling` e `sklearn.cluster` do *software* Python contendo as funções `KMeansSMOTE()` e `KMeans()`, respectivamente, foram utilizadas.

Cinco hiperparâmetros foram importantes ao aplicar o kmeans-SMOTE. Quatro deles são parâmetros do próprio algoritmo `KMeansSMOTE()`, e o hiperparâmetro "`n_clusters`" é do algoritmo `KMeans()`, o qual é utilizado na etapa de agrupamento. Em relação ao `KMeansSMOTE()`, o hiperparâmetro chamado "`sampling_strategy`"é usado para controlar a estratégia de amostragem, ou seja, a proporção do tamanho da classe minoritária em relação à majoritária. Já o '`k_neighbors`" é usado para controlar o número de vizinhos próximos a serem considerados ao gerar instâncias sintéticas para a classe minoritária.

"cluster_balance_threshold" controla o limite de equilíbrio entre os *clusters* identificados pelo algoritmo `KMeans()`, e é usado para determinar se os clusters têm um tamanho suficientemente equilibrado antes de realizar a geração de instâncias sintéticas. O hiperparâmetro "density_exponent" controla a ponderação da densidade ao selecionar vizinhos para a geração de instâncias sintéticas, usado para equilibrar a importância das instâncias sintéticas em regiões mais densas ou mais escassas do espaço de recursos. Por fim, o hiperparâmetro "n_clusters" é usado para controlar o número de clusters a serem formados, assim como o número de centróides a serem gerados no algoritmo `KMeans()`.

Testamos a técnica kmeans-SMOTE para diferentes combinações de valores de "sampling_strategy", "k_neighbors", "n_clusters", "cluster_balance_threshold" e "density_exponent". Dessa forma, para cada combinação, ajustamos um modelo de Regressão Logística, determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos as métricas de desempenho no conjunto de teste a fim de determinar os valores mais adequados para esses cinco parâmetros.

Escolhemos a melhor combinação com base na métrica F1-score e a disponibilizamos na Tabela 4.7. Em caso de empate, ordenamos com base na métrica AUC-ROC.

Tabela 4.7: Tabela com a melhor combinação de "sampling_strategy", "k_neighbors", "n_clusters", "cluster_balance_threshold" e "density_exponent" classificada com base na métrica F1-score.

Base de Dados	"sampling_strategy"	"k_neighbors"	"n_clusters"	"cluster_balance_threshold"	"density_exponent"	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,0574	3	80	0,0014	3	0,8900	0,8603	0,7905	0,8239	0,8952
Renda do Censo	0,4000	6	96	0,0010	2	0,3200	0,5743	0,6144	0,5937	0,7350

Observamos que, para a base de fraude, o parâmetro "sampling_strategy" é igual a 0,0574, o parâmetro "k_neighbors" é igual a 3, o parâmetro "n_clusters" é igual a 80, o parâmetro "cluster_balance_threshold" é igual a 0,0014 e, por fim, o parâmetro "density_exponent" é igual a 3. Sendo assim, o tamanho da classe minoritária é 5,74% do tamanho da majoritária, sendo que a proporção anterior foi de 0,1725%. Para a base de renda do censo, obtemos "sampling_strategy" igual a 0,4000, "k_neighbors" igual a 6, "n_clusters" igual a 96, "cluster_balance_threshold" igual a 0,0010 e, por fim, o parâmetro "density_exponent" igual a 2. Portanto, nesse caso, o tamanho da classe minoritária é 40% do tamanho da majoritária, sendo que proporção anterior foi de 31,72%. Os resultados das métricas calculadas, novamente, mostram que o modelo de fraude está apresentando um desempenho bom, uma vez que todas estão razoavelmente elevadas, e

que o modelo de renda do censo apresenta um desempenho moderado, já que as métricas ainda permanecem abaixo de 0,74.

4.6 ADASYN

Aplicamos o ADASYN nos conjuntos de dados de treinamento, o qual ajusta adaptativamente a distribuição de instâncias sintéticas com base na taxa de densidade de observações da classe minoritária, gerando mais instâncias sintéticas para observações da classe minoritária que são mais difíceis de aprender. Para isso, a biblioteca `imblearn.over_sampling` do *software* Python foi utilizada.

Dois hiperparâmetros foram importantes ao aplicar o ADASYN. O hiperparâmetro chamado "`sampling_strategy`", usado para controlar a estratégia de amostragem que o algoritmo irá adotar, ou seja, a proporção do tamanho da classe minoritária em relação à majoritária. Já o '`n_neighbors`" é usado para controlar o número de vizinhos próximos a serem considerados ao gerar instâncias sintéticas para a classe minoritária.

Testamos a técnica kmeans-SMOTE para diferentes combinações de valores de "`sampling_strategy`" e "`n_neighbors`". Dessa forma, para cada combinação, ajustamos um modelo de Regressão Logística, determinamos o ponto de corte ótimo, o utilizamos para classificar novas observações como pertencentes às classes positiva ou negativa e, a partir disso, calculamos as métricas de desempenho no conjunto de teste a fim de determinar o valore mais adequado para cada um desses dois parâmetros.

A partir disso, escolhemos a melhor combinação com base na métrica F1-score e a disponibilizamos na Tabela 4.8. Em caso de empate, ordenamos com base na métrica AUC-ROC.

Tabela 4.8: Tabela com a melhor combinação de "`sampling_strategy`" e "`n_neighbors`" classificada com base na métrica F1-score.

Base de Dados	"sampling_strategy"	"n_neighbors"	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Fraude	0,0339	5	0,9300	0,8667	0,7905	0,8269	0,8952
Renda do Censo	0,4036	19	0,3300	0,5437	0,6565	0,5948	0,7409

Com base na Tabela 4.8, para a base de fraude, temos o parâmetro "`sampling_strategy`" igual a 0,0339 e o parâmetro "`n_neighbors`" igual a 5. Sendo assim, o tamanho da classe minoritária é 3,39% do tamanho da majoritária, sendo que a proporção anterior foi de 0,1725%. Para a base de renda do censo, obtemos "`sampling_strategy`" igual a 0,4036

e "`n_neighbors`" igual a 19. Portanto, nesse caso, o tamanho da classe minoritária é 40,36% do tamanho da majoritária, sendo que, anteriormente, foi de 31,72%. Em geral, os resultados mostram que o modelo de fraude está apresentando um desempenho razoavelmente bom, mas o modelo de renda do censo apresenta um desempenho moderado, já que as métricas ainda permanecem abaixo de 0,75.

4.7 Comparações

Em resumo, comparando a aplicação de um modelo de Regressão Logística direto na base desbalanceada de fraude e de outro aplicado após a técnica do SMOTE e suas variantes, obtivemos as métricas presentes na Tabela 4.9.

Tabela 4.9: Tabela com métricas de desempenho, do modelo de Regressão Logística ajustado com e sem o SMOTE e suas variantes, aplicadas no conjunto de teste da base de fraude.

Técnica	Ponto de Corte Ótimo	Precisão	Recall	F1-score	AUC-ROC
Sem SMOTE	0,8400	0,8544	0,5946	0,7012	0,7972
SMOTE	0,9100	0,8667	0,7905	0,8269	0,8952
Borderline-SMOTE	0,9700	0,8657	0,7838	0,8227	0,8918
SMOTE-Tomek	0,6100	0,8571	0,7297	0,7883	0,8648
Kmeans-SMOTE	0,8900	0,8603	0,7905	0,8239	0,8952
ADASYN	0,9300	0,8667	0,7905	0,8269	0,8952

Por meio da Tabela 4.9, observamos que a introdução das técnicas de sobreamostragem, sendo elas o SMOTE e suas variantes, resultou, de forma geral, em melhorias significativas nas métricas de desempenho do modelo de Regressão Logística. Em comparação com o modelo sem SMOTE, O Recall apresentou um aumento notável, indicando que o modelo está identificando uma porcentagem ainda maior das transações fraudulentas presentes na base de dados e está conseguindo recuperar uma proporção maior de casos positivos. A Precisão se manteve em níveis aceitáveis ou melhorou em muitos casos, sugerindo que as técnicas de sobreamostragem não comprometeram a Precisão do modelo e sua capacidade de classificar as transações fraudulentas corretamente e reduzir os Falsos Positivos. O F1-score também melhorou, o que indica que o equilíbrio entre Precisão e Recall melhorou substancialmente, sugerindo que o modelo está alcançando um desempenho mais equilibrado na classificação das transações. Por fim, a AUC-ROC, que representa a capacidade geral de discriminação do modelo, também mostrou melhorias em todos os

casos.

Comparando as técnicas entre si, observamos que todas possuem resultados muito próximos para todas as métricas. Notamos também que tanto o SMOTE quanto o ADASYN ofereceram o mesmo desempenho, sendo ele superior às demais técnicas para todas as métricas. Se considerássemos o desempenho no F1-score, uma vez que julgamos essa métrica mais interessante para o nosso problema, já que o F1-score é, particularmente, relevante para um cenário de detecção de fraudes, em que é importante encontrar um compromisso entre a identificação de fraudes verdadeiras e a minimização de Falsos Positivos, oferecendo um balanço entre Precisão e Recall, tanto o SMOTE quanto o ADASYN possuem um F1-score de 0,8269.

Em resumo, a Tabela 4.9 nos mostra que a aplicação do SMOTE e das variantes levou a uma melhoria significativa no desempenho do modelo de Regressão Logística na detecção de fraudes em transações de cartão de crédito, e pode ser benéfica para lidar com conjuntos de dados severamente desbalanceados ao treinar modelos de Regressão Logística, melhorando a capacidade do modelo de generalizar para casos da classe minoritária.

Em seguida, comparando a aplicação de um modelo de Regressão Logística direto na base desbalanceada de renda do censo e de outro aplicado após a técnica do SMOTE e suas variantes, obtivemos as métricas presentes na Tabela 4.10.

Tabela 4.10: Tabela com métricas de desempenho, do modelo de Regressão Logística ajustado com e sem o SMOTE e suas variantes, aplicadas no conjunto de teste da base de renda do censo.

Técnica	**Ponto de Corte Ótimo**	**Precisão**	**Recall**	**F1-score**	**AUC-ROC**
Sem SMOTE	0,3000	0,5572	0,6314	0,5920	0,7361
SMOTE	0,5000	0,5445	0,6582	0,5960	0,7418
Borderline-SMOTE	0,3700	0,5450	0,6539	0,5945	0,7404
SMOTE-Tomek	0,3900	0,5592	0,6301	0,5926	0,7363
Kmeans-SMOTE	0,3200	0,5743	0,6144	0,5937	0,7350
ADASYN	0,3300	0,5437	0,6565	0,5948	0,7409

Já por meio da Tabela 4.10, observamos que a introdução das técnicas de sobreamostragem, sendo elas o SMOTE e suas variantes, não apresentou grandes mudanças nas métricas de desempenho em relação ao modelo de Regressão Logística. A melhora foi ínfima ou nula para todas as técnicas em comparação com o modelo sem balanceamento. Comparando as técnicas entre si, observamos que todas possuem resultados muito próximos para todas as métricas.

Em resumo, a Tabela 4.10 nos mostra que a aplicação do SMOTE e das variantes não levou a melhorias significativas no desempenho do modelo de Regressão Logística na detecção de indivíduos com renda superior a $50 mil por ano e, portanto, não julgamos uma abordagem benéfica para lidar com conjuntos de dados que não possuem um desbalanceamento tão acentuado ao treinar modelos de Regressão Logística.

Em geral, é importante ressaltar que outros modelos de classificação poderiam ser mais adequados ou levar a desempenhos melhores do que a Regressão Logística, assim como outras diferentes combinações de parâmetros poderiam ter melhores resultados. Ademais, a natureza específica do conjunto de dados e dos requisitos do problema em questão também são fatores importantes que influenciam os resultados e conclusões.

Capítulo 5

Considerações Finais

Este estudo abordou a aplicação de técnicas de sobreamostragem, especificamente o SMOTE e as variantes Boderline-SMOTE, SMOTE-Tomek, Kmeans-SMOTE e ADASYN, para o balanceamento de classes para lidar com conjuntos de dados desbalanceados. A análise foi conduzida em duas bases distintas usando modelos de Regressão Logística: uma relacionada a transações fraudulentas em cartões de crédito com desbalanceamento severo e outra referente à previsão de renda anual com um desbalanceamento moderado.

Analisando as métricas de desempenho na detecção de fraudes, notamos melhorias expressivas após a aplicação das técnicas de sobreamostragem. O SMOTE e o ADASYN se destacaram, oferecendo melhorias substanciais no Recall, crucial para identificar transações fraudulentas. A Precisão manteve-se aceitável, indicando que o modelo continuou a classificar corretamente as transações não fraudulentas. O equilíbrio aprimorado entre Precisão e Recall, evidenciado pelo aumento no F1-score, ressalta a eficácia dessas técnicas na detecção de fraudes.

Contrastando com esse cenário positivo, na previsão de renda anual, as melhorias obtidas nas métricas de desempenho foram menos expressivas. Embora todas as técnicas tenham contribuído para um aumento modesto no Recall, a melhora geral foi considerada ínfima. A análise comparativa entre as técnicas revelou resultados bastante próximos, sugerindo que, neste contexto específico, o uso do SMOTE e variantes pode não ser tão impactante quanto em cenários mais desbalanceados.

É crucial destacar que a escolha da técnica de balanceamento deve considerar a natureza do conjunto de dados e os objetivos específicos do problema. A detecção de fraudes em transações de cartão de crédito, cuja classe minoritária é de grande relevância, beneficia-se significativamente do SMOTE e suas variantes. No entanto, para a previsão

de renda anual, cujo desbalanceamento é menos pronunciado, essas técnicas podem não proporcionar ganhos substanciais.

Em conclusão, este estudo ressalta a importância de avaliar cuidadosamente o impacto das técnicas de balanceamento de classes em diferentes contextos. A escolha da abordagem deve ser guiada pela compreensão profunda do conjunto de dados e dos requisitos específicos do problema. Além disso, é fundamental reconhecer que outros modelos e ajustes de parâmetros podem oferecer soluções mais eficazes. Este trabalho contribui para a compreensão do papel do balanceamento de classes na Regressão Logística, fornecendo perspectivas valiosas para aprimorar a eficácia dos modelos em situações específicas de desbalanceamento.

Referências Bibliográficas

Batista, G. E., Bazzan, A. L., Monard, M. C. *et al.* (2003). Balancing training data for automated annotation of keywords: a case study. Em *WOB*, páginas 10–18.

Becker, B. e Kohavi, R. (1996). Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

Chawla, N. V., Bowyer, K. W., Hall, L. O. e Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, **16**, 321–357.

Cloud, S. (2023). How to get the optimal cutoff point of the roc in logistic regression as a number. Disponível em: https://saturncloud.io/blog/how-to-get-the-optimal-cutoff-point-of-the-roc-in-logistic-regression-as-a-numbe#:~:text=To%20determine%20the%20optimal%20cutoff%20point%2C%20we%20can%20use%20the,index%20at%20each%20threshold%20setting. Acesso em: 26 ago. 2023.

Diniz, C. e Louzada, F. (2013). *Métodos Estatísticos para Análise de Dados de Crédito*. 6th Brazilian Conference on Statistical Modelling in Insurance and Finance.

Han, H., Wang, W.-Y. e Mao, B.-H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. Em *Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I 1*, páginas 878–887. Springer.

He, H., Bai, Y., Garcia, E. A. e Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. Em *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, páginas 1322–1328. IEEE.

Hernandez, J., Carrasco-Ochoa, J. A. e Martínez-Trinidad, J. F. (2013). An empirical study of oversampling and undersampling for instance selection methods on imbalance datasets. Em *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 18th Iberoamerican Congress, CIARP 2013, Havana, Cuba, November 20-23, 2013, Proceedings, Part I 18*, páginas 262–269. Springer.

Kaggle (2023). Credit card fraud detection. Disponível em: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud. Acesso em: 04 ago. 2023.

King, G. e Zeng, L. (2001). Logistic regression in rare events data. *Political analysis*, **9**(2), 137–163.

Kovács, G. (2019). An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing*.

Last, F., Douzas, G. e Bacao, F. (2017). Oversampling for imbalanced learning based on k-means and smote, 1–19.

LaValley, M. P. (2008). Logistic regression. *Circulation*, **117**(18), 2395–2399.

Tomek, I. (1976). Two modifications of cnn. *IEEE Transactions on Systems, Man, and Communications*.

Apêndice A

Códigos

Os códigos feitos no *software* Python e utilizados na parte da aplicação desde TCC seguem abaixo.

A.1 Base de Dados de Fraude

```
########################################
############## BIBLIOTECAS ##############
########################################

import pandas as pd
import seaborn as sns
import numpy as np
import imblearn
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE, ADASYN, BorderlineSMOTE, KMeansSMOTE
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score,
                            roc_auc_score, roc_curve
from collections import Counter

###################################
############## DADOS ##############
###################################

dados = pd.read_csv(r"C:\ESTATISTICA\TRABALHO DE GRADUAÇÃO (TG) 1\Dados\creditcard.csv\creditcard.csv")
dados

################################################
############## ANALISE DESCRITIVA ##############
```

```
################################################

dados.describe()

#### VARIÁVEL RESPOSTA - Visualizando a quantidade de transações fraudulentas e não fraudulentas ####

## Gráfico de Pizza

# Definir os rótulos para o gráfico de pizza
labels = 'Não Fraude', 'Fraude'

# Calcular a contagem de transações fraudulentas e não fraudulentas no conjunto de dados
sizes = [dados.Class[dados['Class']==0].count(), dados.Class[dados['Class']==1].count()]

# Criar uma nova figura e eixo para o gráfico de pizza
fig1, ax1 = plt.subplots(figsize=(10, 6))

# Plotar o gráfico de pizza com os tamanhos e rótulos calculados
ax1.pie(sizes, labels=labels, autopct=lambda p : '{:.4f}%  ({:,.0f})'.format(p,p * sum(sizes)/100),
        shadow=False, startangle=120)

# Configurar o eixo como igual para fazer o gráfico de pizza um círculo perfeito
ax1.axis('equal')

# Definir o título do gráfico de pizza com distância adicionada (pad) entre o título e o gráfico
title = "Porcentagem de Transações Fraudulentas e Não Fraudulentas"
plt.title(title, size=16, pad=20)  # You can change the pad value to adjust the distance

# Mostrar o gráfico de pizza
plt.show()

#### COVARIÁVEL AMOUNT ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['Amount'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['Amount']), max(dados['Amount'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'Amount'
bplot = plt.boxplot(dados['Amount'], vert=1, patch_artist=True)
```

```
# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna Amount', fontsize=18)
plt.xlabel('')
plt.ylabel('Amount')

plt.tight_layout()
plt.show()

dados["Amount"].describe().round(4)

dados.drop(columns=['Class','Amount','Time']).describe().round(4)

################################################################
############### PREPARACAO DO CONJUNTO DE DADOS ###############
################################################################

##### Retirando possíveis NAs do conjunto de dados
dados = dados.dropna(axis=0) # não há NAs

##### Escalonamento da covariável Amount
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler() # O RobustScaler é menos suscetível a valores atípicos

dados['scaled_amount'] = rob_scaler.fit_transform(dados['Amount'].values.reshape(-1,1))

dados.drop(['Amount'], axis=1, inplace=True)

##### Dividindo entre covariáveis e variável resposta
x = dados.drop(columns=['Class'])
y = dados['Class']

##### Dividindo em treino e teste
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y, test_size = 0.30, random_state = 0)
```

```
################################################
############## REGRESSAO LOGISTICA ##############
################################################

#### Regressão Logística

# Treine o modelo de regressão logística
classif_RL = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(x_train, y_train)

# Obtenha as probabilidades previstas
y_prob_test = classif_RL.predict_proba(x_test)[:, 1]

# Encontrar o ponto de corte ótimo para maximizar o F1-score
thresholds = np.arange(0, 1.01, 0.01)
f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
optimal_threshold = thresholds[np.argmax(f1_scores)]

print(f'O ponto de corte ótimo é: {optimal_threshold:.4f}')

# Usando o ponto de corte ótimo para fazer previsões binárias
y_pred = (y_prob_test >= optimal_threshold).astype(int)

# Calcular métricas de desempenho
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_roc_test_sm = roc_auc_score(y_test, y_pred)

print(f'Precisão: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'AUC-ROC Score: {auc_roc_test_sm:.4f}')

###################################
############## SMOTE ##############
###################################

# Defina os valores para sampling_strategy e k_neighbors
strategy = np.linspace(0.005, 1, 1000)
neighbors = np.arange(1, 20)

# Crie um DataFrame para armazenar as métricas
metricas_smote = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):
        # Aplicar o SMOTE
        sm = SMOTE(random_state=0, sampling_strategy=strategy[i], k_neighbors=neighbors[j])
        x_train_sm, y_train_sm = sm.fit_resample(x_train, y_train)
```

```
        # Treinar o modelo de regressão logística
        classif_sm = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_sm, y_train_sm)

        # Obter as probabilidades previstas
        y_prob_test = classif_sm.predict_proba(x_test)[:, 1]

        # Encontrar o ponto de corte ótimo para maximizar o F1-score
        thresholds = np.arange(0, 1.01, 0.01)
        f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
        optimal_threshold = thresholds[np.argmax(f1_scores)]

        # Usar o ponto de corte ótimo para fazer previsões binárias
        y_pred = (y_prob_test >= optimal_threshold).astype(int)

        # Calcular métricas de desempenho
        precision_test_sm = precision_score(y_test, y_pred)
        recall_test_sm = recall_score(y_test, y_pred)
        f1_test_sm = f1_score(y_test, y_pred)
        auc_roc_test_sm = roc_auc_score(y_test, y_pred)

        # Criar um DataFrame com as métricas
        metricas_new = pd.DataFrame([[strategy[i],neighbors[j],optimal_threshold.round(4),precision_test_sm.round(4),
                                    recall_test_sm.round(4), f1_test_sm.round(4), auc_roc_test_sm.round(4)]],
                                    columns=["sampling_strategy", "k_neighbors", "Ponto de corte ótimo", "Precisão",
                                    "Recall", "F1-score", "AUC-ROC"])
        metricas_smote = pd.concat([metricas_smote, metricas_new], ignore_index=True)

        l = l + 1
        print(l)

print(metricas_smote)

#### As  melhores combinações de sampling_strategy e k_neighbors classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_smote.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

################################################
############## Borderline SMOTE ###############
################################################

#### Vendo o melhor valor pra sampling_strategy, k_neighbors e m_neighbors ####
strategy = np.linspace(0.005, 1, 100)
neighbors_k = np.arange(1, 10)
neighbors_m = np.arange(1, 10)

metricas_borderline = pd.DataFrame()

l = 0
for i in range(len(strategy)):
```

```
    for j in range(len(neighbors_k)):
        for k in range(len(neighbors_m)):

            ## Aplicando o Borderline SMOTE
            borderline = BorderlineSMOTE(random_state = 0,  sampling_strategy = strategy[i], k_neighbors = neighbors_k[j],
                                         m_neighbors = neighbors_m[k], kind='borderline-1')
            x_train_borderline, y_train_borderline = borderline.fit_resample(x_train, y_train)

            # Treine o modelo de regressão logística
            classif_borderline = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(x_train_borderline,
             y_train_borderline)

            ## Obtenha as probabilidades previstas
            y_prob_test = classif_borderline.predict_proba(x_test)[:, 1]

            # Encontrar o ponto de corte ótimo para maximizar o F1-score
            thresholds = np.arange(0, 1.01, 0.01)
            f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
            optimal_threshold = thresholds[np.argmax(f1_scores)]

            # Usar o ponto de corte ótimo para fazer previsões binárias
            y_pred = (y_prob_test >= optimal_threshold).astype(int)

            # Calcular métricas de desempenho
            precision_test_borderline = precision_score(y_test, y_pred)
            recall_test_borderline = recall_score(y_test, y_pred)
            f1_test_borderline = f1_score(y_test, y_pred)
            auc_roc_test_borderline = roc_auc_score(y_test, y_pred)

            # Fazendo a tabela de metricas
            metricas_new = pd.DataFrame([[strategy[i],neighbors_k[j],neighbors_m[k],optimal_threshold.round(4),
                                          precision_test_borderline.round(4),recall_test_borderline.round(4),
                                          f1_test_borderline.round(4),auc_roc_test_borderline.round(4)]],
                                          columns = ["sampling_strategy","k_neighbors","m_neighbors",
                                          "Ponto de corte ótimo","Precisão","Recall","F1-score","AUC-ROC"])
            metricas_borderline = pd.concat([metricas_borderline, metricas_new], ignore_index = True)

            l = l + 1
            print(l)

print(metricas_borderline)

#### As  melhores combinações classificadas com base na métrica F1-score (e como desempate, AUC-ROC) ####
metricas_borderline.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

########################################
############## SMOTE-Tomek ##############
########################################

#### Vendo o melhor valor pra sampling_strategy_smtomek, sampling_strategy_smote e n_neighbors_smote ####
```

```
strategy = np.linspace(0.01, 1, 100) #(0.005, 1, 100)
strategy_smote = np.linspace(0.005, 1, 100) #(0.005, 1, 100)
neighbors_smote = np.arange(1, 10)

metricas_smtomek = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(strategy_smote)):
        for k in range(len(neighbors_smote)):

            ## Aplicando o SMOTE-Tomek
            smtomek = SMOTETomek(random_state = 0, sampling_strategy = strategy[i],
                                 tomek = TomekLinks(sampling_strategy='majority'),
                                 smote = SMOTE(random_state = 0, sampling_strategy = strategy_smote[k],
                                 k_neighbors = neighbors_smote[k]))
            x_train_smtomek, y_train_smtomek = smtomek.fit_resample(x_train, y_train)

            # Treine o modelo de regressão logística
            classif_smtomek = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_smtomek,
            y_train_smtomek)

            # Obtenha as probabilidades previstas
            y_prob_test = classif_smtomek.predict_proba(x_test)[:, 1]

            # Encontrar o ponto de corte ótimo para maximizar o F1-score
            thresholds = np.arange(0, 1.01, 0.01)
            f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
            optimal_threshold = thresholds[np.argmax(f1_scores)]

            # Usar o ponto de corte ótimo para fazer previsões binárias
            y_pred = (y_prob_test >= optimal_threshold).astype(int)

            # Calcular métricas de desempenho
            precision_test_smtomek = precision_score(y_test, y_pred)
            recall_test_smtomek = recall_score(y_test, y_pred)
            f1_test_smtomek = f1_score(y_test, y_pred)
            auc_roc_test_smtomek = roc_auc_score(y_test, y_pred)

            # Fazendo a tabela de metricas
            metricas_new = pd.DataFrame([[strategy[i],strategy_smote[j],neighbors_smote[k],optimal_threshold.round(4),
                                precision_test_smtomek.round(4),recall_test_smtomek.round(4),
                                f1_test_smtomek.round(4),auc_roc_test_smtomek.round(4)]],
                                columns = ["sampling_strategy_smtomek","sampling_strategy_smote",
                                "n_neighbors_smote","Ponto de corte ótimo","Precisão","Recall","F1-score",
                                "AUC-ROC"])
            metricas_smtomek = pd.concat([metricas_smtomek, metricas_new], ignore_index = True)

            l = l + 1
            print(l)
```

```
print(metricas_smtomek)

#### As  melhores combinações classificadas com base na métrica F1-score (e como desempate, AUC-ROC) ####
metricas_smtomek.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

########################################
############## kmeans-SMOTE ##############
########################################

## Vendo o melhor valor pra sampling_strategy, k_neighbors, n_clusters, cluster_balance_threshold e density_exponent ##
strategy = np.linspace(0.005, 1, 20)
neighbors = np.arange(2, 10)
clusters = np.arange(80, 100, 8)
cluster_balance = np.linspace(0.001, 0.005, 20)
density = np.arange(1, 10)

metricas_kmeans_sm = pd.DataFrame()

n = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):
        for k in range(len(clusters)):
            for l in range(len(cluster_balance)):
                for m in range(len(density)):

                    ## Aplicando o kmeans-SMOTE
                    kmeans_sm = KMeansSMOTE(random_state = 0,sampling_strategy = strategy[i],k_neighbors = neighbors[j],
                                            kmeans_estimator = KMeans(random_state = 0, init='k-means++', n_init = 1,
                                            n_clusters = clusters[k]), cluster_balance_threshold = cluster_balance[l],
                                            density_exponent = density[m])
                    x_train_kmeans_sm, y_train_kmeans_sm = kmeans_sm.fit_resample(x_train, y_train)

                    # Treine o modelo de regressão logística
                    classif_kmeans_sm = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(
                    x_train_kmeans_sm, y_train_kmeans_sm)

                    ## Obtenha as probabilidades previstas
                    y_prob_test = classif_kmeans_sm.predict_proba(x_test)[:, 1]

                    # Encontrar o ponto de corte ótimo para maximizar o F1-score
                    thresholds = np.arange(0, 1.01, 0.01)
                    f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
                    optimal_threshold = thresholds[np.argmax(f1_scores)]

                    # Usar o ponto de corte ótimo para fazer previsões binárias
                    y_pred = (y_prob_test >= optimal_threshold).astype(int)

                    # Calcular métricas de desempenho
                    precision_test_kmeans_sm = precision_score(y_test, y_pred)
```

```
                    recall_test_kmeans_sm = recall_score(y_test, y_pred)
                    f1_test_kmeans_sm = f1_score(y_test, y_pred)
                    auc_roc_test_kmeans_sm = roc_auc_score(y_test, y_pred)

                    # Fazendo a tabela de metricas
                    metricas_new = pd.DataFrame([[strategy[i],neighbors[j],clusters[k],cluster_balance[l],density[m],
                                                  optimal_threshold.round(4),precision_test_kmeans_sm.round(4),
                                                  recall_test_kmeans_sm.round(4),f1_test_kmeans_sm.round(4),
                                                  auc_roc_test_kmeans_sm.round(4)]],
                                                  columns = ["sampling_strategy","k_neighbors","n_clusters",
                                                  "cluster_balance_threshold","density_exponent","Ponto de corte ótimo",
                                                  "Precisão","Recall","F1-score","AUC-ROC"])
                    metricas_kmeans_sm = pd.concat([metricas_kmeans_sm, metricas_new], ignore_index = True)

                    n = n + 1
                    print(n)

print(metricas_kmeans_sm)

#### As  melhores combinações classificadas com base na métrica F1-score (e como desempate, AUC-ROC) ####
metricas_kmeans_sm.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

####################################
############## ADASYN ##############
####################################

#### Vendo o melhor valor pra sampling_strategy e n_neighbors ####
strategy = np.linspace(0.005, 1, 1000)
neighbors = np.arange(1, 20)

metricas_adasyn = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):

        ## Aplicando o ADASYN
        adasyn = ADASYN(random_state = 0, sampling_strategy = strategy[i], n_neighbors = neighbors[j])
        x_train_adasyn, y_train_adasyn = adasyn.fit_resample(x_train, y_train)

        # Treine o modelo de regressão logística
        classif_adasyn = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_adasyn,
        y_train_adasyn)

        # Obtenha as probabilidades previstas
        y_prob_test = classif_adasyn.predict_proba(x_test)[:, 1]

        # Encontrar o ponto de corte ótimo para maximizar o F1-score
        thresholds = np.arange(0, 1.01, 0.01)
        f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
```

```
        optimal_threshold = thresholds[np.argmax(f1_scores)]

        # Usar o ponto de corte ótimo para fazer previsões binárias
        y_pred = (y_prob_test >= optimal_threshold).astype(int)

        # Calcular métricas de desempenho
        precision_test_adasyn = precision_score(y_test, y_pred)
        recall_test_adasyn = recall_score(y_test, y_pred)
        f1_test_adasyn = f1_score(y_test, y_pred)
        auc_roc_test_adasyn = roc_auc_score(y_test, y_pred)

        # Fazendo a tabela de metricas
        metricas_new = pd.DataFrame([[strategy[i],neighbors[j],optimal_threshold.round(4),precision_test_adasyn.round(4),
                    recall_test_adasyn.round(4),f1_test_adasyn.round(4),auc_roc_test_adasyn.round(4)]],
                    columns = ["sampling_strategy","n_neighbors","Ponto de corte ótimo","Precisão",
                    "Recall","F1-score","AUC-ROC"])
        metricas_adasyn = pd.concat([metricas_adasyn, metricas_new], ignore_index = True)

        l = l + 1
        print(l)

print(metricas_adasyn)

#### As  melhores combinações de sampling_strategy e k_neighbors classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_adasyn.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)
```

A.2 Base de Dados de Renda do Censo

```
########################################
############## BIBLIOTECAS ##############
########################################

import pandas as pd
import seaborn as sns
import numpy as np
import imblearn
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE, ADASYN, BorderlineSMOTE, KMeansSMOTE
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score,
                            roc_auc_score, roc_curve
from collections import Counter

###################################
```

```
############## DADOS ##############
###################################

dados = pd.read_csv(r"C:\ESTATISTICA\TRABALHO DE GRADUAÇÃO (TG) 1\Dados\adult\adult.data",
                    sep=",",
                    header=None,
                    names=['age','workclass','fnlwgt','education','education-num','marital-status',
                           'occupation','relationship','race','sex','capital-gain','capital-loss',
                           'hours-per-week','native-country','income'])
dados

################################################
############## ANALISE DESCRITIVA ##############
################################################

dados.describe()

# Mapeia '<=50K' para 0 e '>50K' para 1
dados['income'] = dados['income'].map({' <=50K': 0, ' >50K': 1})

# Renomeia algumas colunas
dados.rename(columns={'education-num': 'education_num'}, inplace=True)
dados.rename(columns={'capital-gain': 'capital_gain'}, inplace=True)
dados.rename(columns={'capital-loss': 'capital_loss'}, inplace=True)
dados.rename(columns={'hours-per-week': 'hours_per_week'}, inplace=True)

# Seleciona apenas as colunas continuas
dados = dados[['age', 'fnlwgt','education_num','capital_gain','capital_loss','hours_per_week','income']]

#### VARIÁVEL RESPOSTA - Visualizando a quantidade de transações fraudulentas e não fraudulentas ####

## Gráfico de Pizza

# Definir os rótulos para o gráfico de pizza
labels = '<=50K', '>50K'

# Calcular a contagem de transações fraudulentas e não fraudulentas no conjunto de dados
sizes = [dados.income[dados['income']==0].count(), dados.income[dados['income']==1].count()]

# Criar uma nova figura e eixo para o gráfico de pizza
fig1, ax1 = plt.subplots(figsize=(10, 6))

# Plotar o gráfico de pizza com os tamanhos e rótulos calculados
ax1.pie(sizes, labels=labels, autopct=lambda p : '{:.4f}%  ({:,.0f})'.format(p,p * sum(sizes)/100), shadow=False,
        startangle=120)

# Configurar o eixo como igual para fazer o gráfico de pizza um círculo perfeito
ax1.axis('equal')

# Definir o título do gráfico de pizza com distância adicionada (pad) entre o título e o gráfico
```

```
title = "Porcentagem de Renda <=50K e >50K"
plt.title(title, size=16, pad=20)  # You can change the pad value to adjust the distance

# Mostrar o gráfico de pizza
plt.show()

#### COVARIÁVEL Age ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['age'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['age']), max(dados['age'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'Age'
bplot = plt.boxplot(dados['age'], vert=1, patch_artist=True)

# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna age', fontsize=18)
plt.xlabel('')
plt.ylabel('age')

plt.tight_layout()
plt.show()

dados["age"].describe().round(4)
```

```
#### COVARIÁVEL fnlwgt ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['fnlwgt'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['fnlwgt']), max(dados['fnlwgt'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'fnlwgt'
bplot = plt.boxplot(dados['fnlwgt'], vert=1, patch_artist=True)

# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna fnlwgt', fontsize=18)
plt.xlabel('')
plt.ylabel('fnlwgt')

plt.tight_layout()
plt.show()

dados["fnlwgt"].describe().round(4)

#### COVARIÁVEL education_num ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))
```

```
sns.distplot(dados['education_num'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['education_num']), max(dados['education_num'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'education_num'
bplot = plt.boxplot(dados['education_num'], vert=1, patch_artist=True)

# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna education_num', fontsize=18)
plt.xlabel('')
plt.ylabel('education_num')

plt.tight_layout()
plt.show()

dados["education_num"].describe().round(4)

#### COVARIÁVEL capital_gain ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['capital_gain'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['capital_gain']), max(dados['capital_gain'])])

plt.show()
```

```
## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'capital_gain'
bplot = plt.boxplot(dados['capital_gain'], vert=1, patch_artist=True)

# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna capital_gain', fontsize=18)
plt.xlabel('')
plt.ylabel('capital_gain')

plt.tight_layout()
plt.show()

dados["capital_gain"].describe().round(4)

#### COVARIÁVEL fnlwgt ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['capital_loss'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['capital_loss']), max(dados['capital_loss'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'capital_loss'
bplot = plt.boxplot(dados['capital_loss'], vert=1, patch_artist=True)
```

```
# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)

# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna capital_loss', fontsize=18)
plt.xlabel('')
plt.ylabel('capital_loss')

plt.tight_layout()
plt.show()

dados["capital_loss"].describe().round(4)

#### COVARIÁVEL hours_per_week ####

## Histograma normalizado com uma estimativa de densidade de kernel sobreposta

fig, ax = plt.subplots(figsize=(10, 6))

sns.distplot(dados['hours_per_week'], ax=ax, color='royalblue')
ax.set_title('', fontsize=14)
ax.set_xlim([min(dados['hours_per_week']), max(dados['hours_per_week'])])

plt.show()

## Boxplot

plt.figure(figsize=(8, 6))

# Plotando o boxplot da coluna 'hours_per_week'
bplot = plt.boxplot(dados['hours_per_week'], vert=1, patch_artist=True)

# Definindo a cor da caixa para "lightblue"
box_color = 'royalblue'
bplot['boxes'][0].set(color='royalblue', linewidth=1)
bplot['boxes'][0].set(facecolor=box_color)
```

```
# Definindo a cor das linhas de contorno
for whisker in bplot['whiskers']:
    whisker.set(color='black', linewidth=1)

for cap in bplot['caps']:
    cap.set(color='black', linewidth=1)

for median in bplot['medians']:
    median.set(color='black', linewidth=1)

# Adicionando título e rótulos aos eixos
plt.title('Boxplot da Coluna hours_per_week', fontsize=18)
plt.xlabel('')
plt.ylabel('hours_per_week')

plt.tight_layout()
plt.show()

dados["hours_per_week"].describe().round(4)

################################################################
############## PREPARACAO DO CONJUNTO DE DADOS ##############
################################################################

##### Retirando possíveis NAs do conjunto de dados
dados = dados.dropna(axis=0) # não há NAs

dados.dtypes

##### Escalonando as covariáveis necessárias
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler() # O RobustScaler é menos suscetível a valores atípicos

dados['scaled_age'] = rob_scaler.fit_transform(dados['age'].values.reshape(-1,1))
dados['scaled_fnlwgt'] = rob_scaler.fit_transform(dados['fnlwgt'].values.reshape(-1,1))
dados['scaled_education_num'] = rob_scaler.fit_transform(dados['education_num'].values.reshape(-1,1))
dados['scaled_capital_gain'] = rob_scaler.fit_transform(dados['capital_gain'].values.reshape(-1,1))
dados['scaled_capital_loss'] = rob_scaler.fit_transform(dados['capital_loss'].values.reshape(-1,1))
dados['scaled_hours_per_week'] = rob_scaler.fit_transform(dados['hours_per_week'].values.reshape(-1,1))

dados.drop(['age','fnlwgt','education_num','capital_gain','capital_loss','hours_per_week'], axis=1, inplace=True)

##### Dividindo entre covariáveis e variável resposta
x = dados.drop(columns=['income'])
y = dados['income']

##### Dividindo em treino e teste
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y, test_size = 0.30, random_state = 0)
```

```
################################################
############## REGRESSAO LOGISTICA ##############
################################################

#### Regressão Logística

# Treine o modelo de regressão logística
classif_RL = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(x_train, y_train)

# Obtenha as probabilidades previstas
y_prob_test = classif_RL.predict_proba(x_test)[:, 1]

# Encontrar o ponto de corte ótimo para maximizar o F1-score
thresholds = np.arange(0, 1.01, 0.01)
f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
optimal_threshold = thresholds[np.argmax(f1_scores)]

print(f'O ponto de corte ótimo é: {optimal_threshold:.4f}')

# Usando o ponto de corte ótimo para fazer previsões binárias
y_pred = (y_prob_test >= optimal_threshold).astype(int)

# Calcular métricas de desempenho
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc_roc_test_sm = roc_auc_score(y_test, y_pred)

print(f'Precisão: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'AUC-ROC Score: {auc_roc_test_sm:.4f}')

###################################
############## SMOTE ##############
###################################

# Defina os valores para sampling_strategy e k_neighbors
strategy = np.linspace(0.5, 1, 1000)
neighbors = np.arange(1, 20)

# Crie um DataFrame para armazenar as métricas
metricas_smote = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):
        # Aplicar o SMOTE
        sm = SMOTE(random_state=0, sampling_strategy=strategy[i], k_neighbors=neighbors[j])
        x_train_sm, y_train_sm = sm.fit_resample(x_train, y_train)
```

```
        # Treinar o modelo de regressão logística
        classif_sm = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_sm, y_train_sm)

        # Obter as probabilidades previstas
        y_prob_test = classif_sm.predict_proba(x_test)[:, 1]

        # Encontrar o ponto de corte ótimo para maximizar o F1-score
        thresholds = np.arange(0, 1.01, 0.01)
        f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
        optimal_threshold = thresholds[np.argmax(f1_scores)]

        # Usar o ponto de corte ótimo para fazer previsões binárias
        y_pred = (y_prob_test >= optimal_threshold).astype(int)

        # Calcular métricas de desempenho
        precision_test_sm = precision_score(y_test, y_pred)
        recall_test_sm = recall_score(y_test, y_pred)
        f1_test_sm = f1_score(y_test, y_pred)
        auc_roc_test_sm = roc_auc_score(y_test, y_pred)

        # Criar um DataFrame com as métricas
        metricas_new = pd.DataFrame([[strategy[i],neighbors[j],optimal_threshold.round(4),precision_test_sm.round(4),
                                    recall_test_sm.round(4), f1_test_sm.round(4), auc_roc_test_sm.round(4)]],
                                    columns=["sampling_strategy", "k_neighbors", "Ponto de corte ótimo", "Precisão",
                                    "Recall", "F1-score", "AUC-ROC"])
        metricas_smote = pd.concat([metricas_smote, metricas_new], ignore_index=True)

        l = l + 1
        print(l)
print(metricas_smote)

#### As  melhores combinações de sampling_strategy e k_neighbors classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_smote.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

################################################
############## Borderline SMOTE ###############
################################################

#### Vendo o melhor valor pra sampling_strategy, k_neighbors e m_neighbors ####
strategy = np.linspace(0.4, 1, 100)
neighbors_k = np.arange(1, 10)
neighbors_m = np.arange(1, 10)

metricas_borderline = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(neighbors_k)):
```

```
        for k in range(len(neighbors_m)):

            ## Aplicando o Borderline SMOTE
            borderline = BorderlineSMOTE(random_state = 0,  sampling_strategy = strategy[i], k_neighbors = neighbors_k[j],
                                         m_neighbors = neighbors_m[k], kind='borderline-1')
            x_train_borderline, y_train_borderline = borderline.fit_resample(x_train, y_train)

            # Treine o modelo de regressão logística
            classif_borderline = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(x_train_borderline,
            y_train_borderline)

            ## Obtenha as probabilidades previstas
            y_prob_test = classif_borderline.predict_proba(x_test)[:, 1]

            # Encontrar o ponto de corte ótimo para maximizar o F1-score
            thresholds = np.arange(0, 1.01, 0.01)
            f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
            optimal_threshold = thresholds[np.argmax(f1_scores)]

            # Usar o ponto de corte ótimo para fazer previsões binárias
            y_pred = (y_prob_test >= optimal_threshold).astype(int)

            # Calcular métricas de desempenho
            precision_test_borderline = precision_score(y_test, y_pred)
            recall_test_borderline = recall_score(y_test, y_pred)
            f1_test_borderline = f1_score(y_test, y_pred)
            auc_roc_test_borderline = roc_auc_score(y_test, y_pred)

            # Fazendo a tabela de metricas
            metricas_new = pd.DataFrame([[strategy[i],neighbors_k[j],neighbors_m[k],optimal_threshold.round(4),
                                         precision_test_borderline.round(4),recall_test_borderline.round(4),
                                         f1_test_borderline.round(4),auc_roc_test_borderline.round(4)]],
                                         columns = ["sampling_strategy","k_neighbors","m_neighbors",
                                         "Ponto de corte ótimo","Precisão","Recall","F1-score","AUC-ROC"])
            metricas_borderline = pd.concat([metricas_borderline, metricas_new], ignore_index = True)

            l = l + 1
            print(l)

print(metricas_borderline)

#### As  melhores combinações classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_borderline.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

########################################
############## SMOTE-Tomek ##############
########################################

#### Vendo o melhor valor pra sampling_strategy_smtomek, sampling_strategy_smote e n_neighbors_smote ####
```

```
strategy = np.linspace(0.4, 1, 100) #(0.005, 1, 100)
strategy_smote = np.linspace(0.4, 1, 100) #(0.005, 1, 100)
neighbors_smote = np.arange(1, 10)

metricas_smtomek = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(strategy_smote)):
        for k in range(len(neighbors_smote)):

            ## Aplicando o SMOTE-Tomek
            smtomek = SMOTETomek(random_state = 0, sampling_strategy = strategy[i],
                                 tomek = TomekLinks(sampling_strategy='majority'),
                                 smote = SMOTE(random_state = 0, sampling_strategy = strategy_smote[k],
                                 k_neighbors = neighbors_smote[k]))
            x_train_smtomek, y_train_smtomek = smtomek.fit_resample(x_train, y_train)

            # Treine o modelo de regressão logística
            classif_smtomek = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_smtomek,
            y_train_smtomek)

            # Obtenha as probabilidades previstas
            y_prob_test = classif_smtomek.predict_proba(x_test)[:, 1]

            # Encontrar o ponto de corte ótimo para maximizar o F1-score
            thresholds = np.arange(0, 1.01, 0.01)
            f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
            optimal_threshold = thresholds[np.argmax(f1_scores)]

            # Usar o ponto de corte ótimo para fazer previsões binárias
            y_pred = (y_prob_test >= optimal_threshold).astype(int)

            # Calcular métricas de desempenho
            precision_test_smtomek = precision_score(y_test, y_pred)
            recall_test_smtomek = recall_score(y_test, y_pred)
            f1_test_smtomek = f1_score(y_test, y_pred)
            auc_roc_test_smtomek = roc_auc_score(y_test, y_pred)

            # Fazendo a tabela de metricas
            metricas_new = pd.DataFrame([[strategy[i],strategy_smote[j],neighbors_smote[k],optimal_threshold.round(4),
                                precision_test_smtomek.round(4),recall_test_smtomek.round(4),
                                f1_test_smtomek.round(4),auc_roc_test_smtomek.round(4)]],
                                columns = ["sampling_strategy_smtomek","sampling_strategy_smote",
                                "n_neighbors_smote","Ponto de corte ótimo","Precisão","Recall","F1-score",
                                "AUC-ROC"])
            metricas_smtomek = pd.concat([metricas_smtomek, metricas_new], ignore_index = True)

            l = l + 1
            print(l)
```

```
print(metricas_smtomek)

#### As  melhores combinações de sampling_strategy e k_neighbors classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_smtomek.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

##########################################
############## kmeans-SMOTE ##############
##########################################

## Vendo o melhor valor pra sampling_strategy, k_neighbors, n_clusters, cluster_balance_threshold e density_exponent ##
strategy = np.linspace(0.4, 1, 10)
neighbors = np.arange(2, 10)
clusters = np.arange(80, 100, 8)
cluster_balance = np.linspace(0.001, 0.005, 10)
density = np.arange(1, 10)

metricas_kmeans_sm = pd.DataFrame()

n = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):
        for k in range(len(clusters)):
            for l in range(len(cluster_balance)):
                for m in range(len(density)):

                    ## Aplicando o kmeans-SMOTE
                    kmeans_sm = KMeansSMOTE(random_state = 0,sampling_strategy = strategy[i],k_neighbors = neighbors[j],
                                            kmeans_estimator = KMeans(random_state = 0, init='k-means++', n_init = 1,
                                            n_clusters = clusters[k]), cluster_balance_threshold = cluster_balance[l],
                                            density_exponent = density[m])
                    x_train_kmeans_sm, y_train_kmeans_sm = kmeans_sm.fit_resample(x_train, y_train)

                    # Treine o modelo de regressão logística
                    classif_kmeans_sm = LogisticRegression(random_state=0, penalty = None, max_iter=1000).fit(
                    x_train_kmeans_sm, y_train_kmeans_sm)

                    ## Obtenha as probabilidades previstas
                    y_prob_test = classif_kmeans_sm.predict_proba(x_test)[:, 1]

                    # Encontrar o ponto de corte ótimo para maximizar o F1-score
                    thresholds = np.arange(0, 1.01, 0.01)
                    f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
                    optimal_threshold = thresholds[np.argmax(f1_scores)]

                    # Usar o ponto de corte ótimo para fazer previsões binárias
                    y_pred = (y_prob_test >= optimal_threshold).astype(int)

                    # Calcular métricas de desempenho
```

```
                    precision_test_kmeans_sm = precision_score(y_test, y_pred)
                    recall_test_kmeans_sm = recall_score(y_test, y_pred)
                    f1_test_kmeans_sm = f1_score(y_test, y_pred)
                    auc_roc_test_kmeans_sm = roc_auc_score(y_test, y_pred)

                    # Fazendo a tabela de metricas
                    metricas_new = pd.DataFrame([[strategy[i],neighbors[j],clusters[k],cluster_balance[l],density[m],
                                                  optimal_threshold.round(4),precision_test_kmeans_sm.round(4),
                                                  recall_test_kmeans_sm.round(4),f1_test_kmeans_sm.round(4),
                                                  auc_roc_test_kmeans_sm.round(4)]],
                                                  columns = ["sampling_strategy","k_neighbors","n_clusters",
                                                  "cluster_balance_threshold","density_exponent","Ponto de corte ótimo",
                                                  "Precisão","Recall","F1-score","AUC-ROC"])
                    metricas_kmeans_sm = pd.concat([metricas_kmeans_sm, metricas_new], ignore_index = True)

                    n = n + 1
                    print(n)

print(metricas_kmeans_sm)

#### As  melhores combinações classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_kmeans_sm.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)

####################################
############## ADASYN ##############
####################################

#### Vendo o melhor valor pra sampling_strategy e n_neighbors ####
strategy = np.linspace(0.4, 1, 1000)
neighbors = np.arange(1, 20)

metricas_adasyn = pd.DataFrame()

l = 0
for i in range(len(strategy)):
    for j in range(len(neighbors)):

        ## Aplicando o ADASYN
        adasyn = ADASYN(random_state = 0, sampling_strategy = strategy[i], n_neighbors = neighbors[j])
        x_train_adasyn, y_train_adasyn = adasyn.fit_resample(x_train, y_train)

        # Treine o modelo de regressão logística
        classif_adasyn = LogisticRegression(random_state=0, penalty=None, max_iter=1000).fit(x_train_adasyn,
        y_train_adasyn)

        # Obtenha as probabilidades previstas
        y_prob_test = classif_adasyn.predict_proba(x_test)[:, 1]

        # Encontrar o ponto de corte ótimo para maximizar o F1-score
```

```
        thresholds = np.arange(0, 1.01, 0.01)
        f1_scores = [f1_score(y_test, (y_prob_test >= t).astype(int)) for t in thresholds]
        optimal_threshold = thresholds[np.argmax(f1_scores)]

        # Usar o ponto de corte ótimo para fazer previsões binárias
        y_pred = (y_prob_test >= optimal_threshold).astype(int)

        # Calcular métricas de desempenho
        precision_test_adasyn = precision_score(y_test, y_pred)
        recall_test_adasyn = recall_score(y_test, y_pred)
        f1_test_adasyn = f1_score(y_test, y_pred)
        auc_roc_test_adasyn = roc_auc_score(y_test, y_pred)

        # Fazendo a tabela de metricas
        metricas_new = pd.DataFrame([[strategy[i],neighbors[j],optimal_threshold.round(4),precision_test_adasyn.round(4),
                                    recall_test_adasyn.round(4),f1_test_adasyn.round(4),auc_roc_test_adasyn.round(4)]],
                                    columns = ["sampling_strategy","n_neighbors","Ponto de corte ótimo","Precisão",
                                    "Recall","F1-score","AUC-ROC"])
        metricas_adasyn = pd.concat([metricas_adasyn, metricas_new], ignore_index = True)

        l = l + 1
        print(l)

print(metricas_adasyn)

#### As  melhores combinações de sampling_strategy e k_neighbors classificadas com base na métrica F1-score
(e como desempate, AUC-ROC) ####
metricas_adasyn.sort_values(by=['F1-score', 'AUC-ROC'], ascending=False)
```