Pontifícia Universidade Católica
do Rio de Janeiro

Fernando Vianna Brasil Medeiros

Aplicação de Aprendizado de Máquinas para Detecção de Imperfeições Geométricas em Vigas

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Engenharia Civil do Departamento de Engenharia Civil e Ambiental da PUC-Rio.

Orientador: Prof. Luiz Carlos Wrobel

Rio de Janeiro
Fevereiro de 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Fernando Vianna Brasil Medeiros

Aplicação de Aprendizado de Máquinas para Detecção de Imperfeições Geométricas em Vigas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Engenharia Civil da PUC-Rio. Aprovada pela Comissão Examinadora abaixo:

Prof. Luiz Carlos Wrobel
Orientador
Departamento de Engenharia Civil e Ambiental – PUC-Rio

Prof. Luiz Fernando Campos Ramos Martha
Departamento de Engenharia Civil e Ambiental - PUC-Rio

Prof. Deane de Mesquita Roehl
Departamento de Engenharia Civil e Ambiental - PUC-Rio

Rio de Janeiro, 28 de Fevereiro de 2023

Fernando Vianna Brasil Medeiros

Graduado em engenharia de Fortificação e Construção pelo Instituto Militar de Engenharia.

Ficha Catalográfica

Vianna Brasil Medeiros, Fernando

Aplicação de Aprendizado de Máquinas para Detecção de Imperfeições Geométricas em Vigas / Fernando Vianna Brasil Medeiros; orientador: Luiz Carlos Wrobel. – 2023.

118 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Civil e Ambiental, 2023.

Inclui bibliografia

1. Estruturas, Ensaios não-destrutivos, Análise dinâmica – Teses. 2. Integridade Estrutural. 3. Detecção de danos. 4. Monitoramento da saúde da estrutura. 5. Inteligência Artificial. I. Wrobel, Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Civil e Ambiental. III. Título.

CDD: 624

À minha família
pelo apoio e encorajamento.

Agradecimentos

Ao meu orientador Professor Luiz Carlos Wrobel pelo estímulo e parceria para a realização deste trabalho.

Ao CNPq e à PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

À minha esposa, Bruna, por todo amor, apoio e compreensão.

Aos meus pais, Elza e Luciano, pela educação, atenção e carinho de todas as horas.

Aos meus sogros, Ulf e Ana Christina, pelo carinho e apoio.

Aos meus chefes e colegas de profissão, pelo apoio.

Aos professores que participaram da Comissão examinadora.

A todos os professores e funcionários do Departamento pelos ensinamentos e pela ajuda.

A todos os amigos e familiares que de uma forma ou de outra me estimularam e ajudaram.

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

Resumo

Vianna Brasil Medeiros, Fernando; Wrobel, Luiz. **Aplicação de Aprendizado de Máquinas para Detecção de Imperfeições Geométricas em Vigas**. Rio de Janeiro, 2023. 118p. Dissertação de Mestrado – Departamento de Engenharia Civil e Ambiental, Pontifícia Universidade Católica do Rio de Janeiro.

O monitoramento da integridade estrutural aumenta de importância dentro do campo de estudo de engenharia civil. Grande parte das cidades dependem de elementos de sua infraestrutura como pontes, barragens e prédios para prover uma série de benefícios para a sociedade moderna. Por outro lado, mesmo o projeto mais conservador não resiste aos efeitos do tempo. Uma boa rotina de manutenção preventiva não exime a necessidade de se ter uma constante verificação e busca de falhas pois em alguns casos isto poderia permitir em catástrofes de grande escala envolvendo grande perda material e até mesmo vidas. Graças ao desenvolvimento tecnológico das últimas décadas foi possível pesquisar e criar ferramentas poderosas que podem ajudar problemas deste tipo. O objetivo desta dissertação é avaliar a aplicação de métodos de Inteligência Artificial na detecção de danos em vigas. A metodologia utiliza parâmetros modais de elementos estruturais para verificar a presença de danos relacionados a redução de rigidez de uma seção transversal. Mais especificamente, os métodos apresentados neste estudo são orientados por dados, então primeiramente o banco de dados para treino e validação dos métodos de IA foi gerado por um programa em *Python* dentro do software de elementos finitos *Abaqus*. Os parâmetrosd modais analisados foram as cinco primeiras frequências naturais das vigas. Foi possível avaliar a performance dos métodos de IA para classificação da presença ou não de danos em diferentes métricas de análise. Por fim, uma comparação paramétrica foi feita entre os modelos de Inteligência Artificial.

Palavras-chave

Integridade Estrutural; Detecção de danos; Monitoramento da saúde da estrutura; Inteligência Artificial.

Abstract

Vianna Brasil Medeiros, Fernando; Wrobel, Luiz (Advisor). **Application of Machine Learning for the Detection of Geometria Imperfection in Beams**. Rio de Janeiro, 2023. 118p. Dissertação de Mestrado – Departamento de Engenharia Civil e Ambiental, Pontifícia Universidade Católica do Rio de Janeiro.

Monitoring structural integrity has become increasingly important in the field of civil engineering. A huge part of cities depend of civil engineer infrastructures such as bridges, dams and buildings to provide several benefits to modern society. On the other hand, even the most conservative design cannot resist the power of time. A good preventive maintenance routine don't let go of the need in constant verification for faults because in some cases that could lead to large scale catastrophes involving big material and life costs. Thanks to technology development over the last decades it was possible to search and create many powerful tools that could help those kind of problems. The objective of this thesis is to assess on the application of Artificial Intelligence Methods to detect damage on beams. The formulation uses modal parameters of a structure to verify the presence of damage related to the reduction of stiffness of a section. More specifically, the methods presented on this study are data-driven, so first a database for training and validating the AI methods were generated in a Python program within the finite element software Abaqus. The modal parameters analyzed were the first five natural frequencies of a beam. It was possible to evaluate the performance of the AI methods when classifying a beam with or without damage on different metrics. Finally, a parametric comparison was made between the Artificial Intelligence methods.

Keywords

Structural Integrity; Damage detection; Structural Health Monitoring; Artificial Intelligence.

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

IA – Inteligência Artificial

SHM – *Structural Health Monitoring*

ML – *Machine Learning*

LINr – *Linear regression*

LOGr – *Logistic regression*

DT – *Decision Trees*

RF – *Random Forests*

SVM – *Support Vector Machines*

MLP – *Multi-layer Perceptron*

RNA – Redes Neurais Artificiais

KNN – *K-nearest neighbors*

Acc – *Accuracy*

CV – *Cross-validation*

SMOTE – *Synthetic Minority Over-sampling Technique*

GI – Ganho de Informação

Não basta fazer coisas boas - é preciso fazê-las bem.

Santo Agostinho.

1
Introdução

1.1
Introdução histórica

A engenharia civil permitiu o desenvolvimento da nossa civilização ao longo da história. Seja por meio de um sistema sanitário ou mesmo na alimentação de água das cidades por aquedutos. Os exemplos mais recentes seriam as grandes hidrelétricas, pontes e edifícios. Apesar dos benefícios evidentes, esses grandes projetos não são eternos e, portanto, as atividades relativas a manutenção e operação não podem ser negligenciadas em face das etapas de concepção e construção.

O monitoramento da integridade de grandes obras se faz lembrar com mais força midiática quando a ruína acontece. Em 15 de dezembro de 1967, a ponte "Silver Bridge" colapsou sobre o rio Ohio durante o carregamento resultante da "hora do rush", resultando na morte de 46 pessoas. O acidente gerou uma comoção nacional e inspirou legisladores a criar regras de inspeções e manutenções regulares em pontes antigas.

Em 1º de agosto de 2007, a ponte "I-35W" também colapsou na hora de maior tráfego, matando 13 pessoas e ferindo outras 145. O que chamou atenção desta vez foi o fato de ela ter sido inspecionada regularmente ao longo dos anos de uso e ter atingido os limites mínimos toleráveis. O fato é que a inspeção visual é limitada e não é adequada para analisar sistemas estruturais mais complexos.

Conforme a norma internacional ISO 16311-1, a recuperação oportuna tem o poder de prolongar consideravelmente a vida útil de uma estrutura. Nesse sentido, diversas técnicas de monitoramento da integridade estrutural foram desenvolvidas ao longo do século XX até os dias de hoje. Métodos baseados em emissões acústicas, ultrassom, vibrações, termografia e inspeção visual são alguns exemplos de campos onde se desenvolveram essas técnicas. Apesar disso, mesmo para um especialista existem situações difíceis de identificar o dano.

Em paralelo, o desenvolvimento tecnológico da velocidade e capacidade de processamento dos computadores permitiu que uma linha de pesquisa crescesse bastante nas últimas décadas: o aprendizado de máquina. Conhecido popularmente como *machine learning*, as técnicas de aprendizado de máquinas

tem como uma de suas grandes vantagens a capacidade de solução de problemas complexos que seriam difíceis de resolver até mesmo por especialistas humanos, e isto parece se encaixar perfeitamente no problema de monitoramento de integridade estrutural. Assim, o número de pesquisas para aplicação de métodos de aprendizado máquina neste sentido está em alta.

1.2 Motivação

Entre os benefícios de ter um controle da integridade estrutural é evidente que o mais importante é evitar a perda de vidas. Entretanto há outros aspectos a serem considerados. Financeiramente, o *feedback* em tempo real permite um planejamento otimizado de intervenções preventivas, o que além de evitar o custo maior que das recuperações corretivas também costumam ser serviços de menor impacto logístico ao usuário, isto é, permite que a estrutura continue operando com menores restrições.

Muito se tem falado de monitoramento da saúde da estrutura em tempo real, ou quase real. Para se ter esse tipo de informação é necessário que o sistema que se deseja monitorar esteja sensoriado adequadamente, que a informação destes sensores seja enviada regularmente para um pós-processamento e o armazenamento da informação para se ter uma análise contínua do comportamento da estrutura.

Desta forma, o desenvolvimento da capacidade de processamento dos computadores é fundamental para permitir este tipo de solução. Campos que exigiriam tempos de processamento que outrora seriam inviáveis tornaram-se populares, como é o caso do uso de técnicas de inteligência artificial.

Como efeito, esta pesquisa é motivada na busca do alinhamento que o desenvolvimento tecnológico permite ao problema de monitoramento de integridade estrutural.

1.3 Objetivo

Objetivos do trabalho:

1. Verificar a viabilidade de utilização de diferentes técnicas de aprendizado de máquinas na identificação de danos em vigas através da utlização de

um parâmetro modal de vibração.

2. Realizar estudo comparativo destas técnicas.

1.4 Estrutura do trabalho

Este trabalho é dividido em sete capítulos. O Capítulo 1 apresenta uma introdução ao tema e objetivos da pesquisa.

No Capítulo 2 será apresentada uma revisão bibliográfica sobre os conceitos de integridade estrutural e análise modal. Serão abordados aspectos teóricos relacionados obtenção de frequências naturais de vibração. Além disso, serão abordados como a comunidade científica tem estudado esses campos.

O Capítulo 3 tem como objetivo apresentar os conceitos teóricos fundamentais dos métodos de *Machine Learning*. Serão apresentados os principais tipos de algoritmos de *Machine Learning*, como regressões, árvores de decisão, redes neurais, entre outros. Além disso, será enfatizada a importância da escolha de um banco de dados adequado e as métricas de análise mais utilizadas. Por fim, serão apresentados alguns estudos sobre a utilização de aprendizado de máquinas no problema de detecção de danos em estruturas.

No Capítulo 4 será apresentada a metodologia adotada para a geração de dados e estruturação dos métodos de *Machine Learning* a serem estudados. Serão descritos os passos necessários para a construção do modelo de dados, que consiste na coleta e processamento dos dados necessários para a aplicação dos métodos de *Machine Learning*. Também serão apresentados os parâmetros que se buscou variar de cada modelo em busca da melhor performance.

No Capítulo 5 serão apresentados os resultados obtidos a partir da aplicação dos modelos de *Machine Learning* construídos na metodologia descrita no Capítulo 4. Serão discutidos os resultados obtidos e as conclusões que podem ser tiradas a partir da análise dos dados gerados pelos modelos.

No Capítulo 6 serão apresentadas as principais conclusões e contribuições do trabalho. Serão destacados os principais resultados obtidos e os pontos fortes e fracos da metodologia adotada. Serão discutidas as limitações do trabalho e sugeridos possíveis caminhos para trabalhos futuros.

Por fim, no Capítulo 7 serão listadas todas as referências bibliográficas utilizadas no trabalho.

2
Revisão bibliográfica

Neste capítulo, serão apresentados de forma breve os conceitos fundamentais sobre integridade estrutural e análise modal, que são essenciais para a compreensão do tema central deste trabalho. A integridade estrutural é um dos principais desafios enfrentados por engenheiros e pesquisadores em todo o mundo, especialmente quando se trata de projetar e manter estruturas complexas e de grande porte, como pontes, edifícios, aeronaves, navios e plataformas offshore. A análise modal, por sua vez, é uma técnica poderosa para caracterizar as propriedades dinâmicas dessas estruturas, permitindo a identificação de suas frequências naturais de vibração, modos de deformação e amortecimento.

Será abordada a identificação modal, que é uma das técnicas mais amplamente utilizadas para avaliar as propriedades dinâmicas das estruturas. Serão apresentados os conceitos básicos de dinâmica estrutural, incluindo as equações diferenciais que descrevem o comportamento dinâmico de uma estrutura e o detalhamento do processo de obtenção das frequências naturais de vibração.

Por fim, será abordado como a comunidade científica tem abordado esses campos, apresentando diversas contribuições da literatura científica.

2.1
Integridade estrutural

A integridade de uma estrutura é um aspecto fundamental que envolve a capacidade de suportar condições de projeto sem falhar. Em função disso, diversas técnicas têm sido desenvolvidas para avaliar a integridade estrutural, sendo os métodos classificados como não-destrutivos amplamente utilizados em todo o mundo. Esses métodos consistem em avaliar o estado da estrutura sem a necessidade de comprometê-la, o que é particularmente vantajoso para minimizar os riscos de perda de materiais e de vidas humanas.

Ao contrário dos métodos destrutivos, que podem ser potencialmente arriscados, a utilização de técnicas não-destrutivas traz inúmeros benefícios, tanto em termos de segurança quanto de aspectos financeiros. Isso porque permitem que a estrutura seja avaliada enquanto em operação, o que reduz a necessidade de interrupções prolongadas ou desligamentos, permitindo a continuidade das operações.

Como o foco principal deste trabalho é a análise modal para o monitoramento

da integridade estrutural, será apresentada uma revisão bibliográfica sobre osa aplicação da análise modal e no contexto de identificação de falhas estruturais.

2.2 Identificação modal

A identificação modal de uma estrutura consiste na obtenção de seus parâmetros modais, são eles: frequências naturais, modos de vibração e amortecimento. Para o presente trabalho as frequências naturais e os modos de vibração serão abordados porém o amortecimento será desprezado.
A equação de viga de Euler-Bernoulli é dada por:

$$\frac{EI}{\rho A}\frac{\partial^4 w(x,t)}{\partial x^4} + \frac{\partial^2 w(x,t)}{\partial t^2} = 0 \tag{2-1}$$

A teoria de vigas de Euler-Bernoulli faz duas considerações: as deformações das seções retas devidas ao cisalhamento são desprezadas; e os efeitos da rotação na deformação da seção são nulos.

As frequências naturais e as formas modais são determinados como se segue.

Utilizando a separação de variáveis $w(x,t) = X(x)T(t)$ procura-se uma solução desta forma. Reescrevendo a equação 2-1 com $\mu = \sqrt{\frac{EI}{\rho A}}$, temos:

$$\mu^2 \frac{X^{(4)}}{X(x)} = -\frac{\ddot{T}(t)}{T(t)} = \omega^2 \tag{2-2}$$

temos, portanto, uma equação dependente apenas da variável tempo:

$$\ddot{T}(t) + \omega^2 T(t) = 0 \tag{2-3}$$

assumindo uma solução da forma:

$$T(t) = asin(\omega t) + bcos(\omega t) \tag{2-4}$$

para a parcela independente do tempo, sendo $\beta^4 = \frac{\omega^2}{\mu^2} = \frac{\rho A \omega^2}{EI}$ temos:

$$X^{(4)}(x) - \beta^4 X(x) = 0 \tag{2-5}$$

supondo soluções da forma De^{sx}, com D e s constantes. Voltando à equação diferencial obtemos:

$$s^4 - \beta^4 = 0 \tag{2-6}$$

seja $s_{1,2} = \beta^2$ temos:

$$\begin{aligned} s_1 &= \beta \\ s_2 &= -\beta \end{aligned} \tag{2-7}$$

e para $s_{3,4}^2 = -\beta^2$, temos:

$$\begin{aligned} s_3 &= j\beta \\ s_4 &= -j\beta \end{aligned} \tag{2-8}$$

logo temos a solução:

$$X(x) = D_1 e^{\beta x} + D_2 e^{-\beta x} + D_3 e^{j\beta x} + D_4 e^{-j\beta x} \tag{2-9}$$

que pode ser escrito da forma:

$$X(x) = C_1 sin\beta x + C_2 cos\beta x + C_3 sinh\beta x + C_4 cosh\beta x \tag{2-10}$$

onde os coeficientes D_1 à D_4, ou C_1 à C_4 são determinados pelas condições de contorno. Para o caso de viga bi-engastada, temos que resolver para n = 1, 2, 3 ... (quantos modos de vibração se queira) a seguinte equação:

$$cos(\beta_n l) cosh(\beta_n l) = 1 \tag{2-11}$$

E assim obtemos as frequências naturais $\omega_n = (\beta_n l)^2 \sqrt{EI/\rho A l^4}$. Para obter as formas modais no caso de viga bi-engastada temos, das condições de contorno, o seguinte:

$$X_n(x) = C_n[(cosh\beta_n x - cos\beta_n x) - (cosh\beta_n x - cos\beta_n x)/(sinh\beta_n x - sin\beta_n x)] = C_n \phi_n x \tag{2-12}$$

Onde ϕ_n são as formas modais.

2.3 Método dos elementos finitos

O método dos elementos finitos foi, sem dúvidas, um das melhores ferramentas desenvolvidas no século XX no contexto de análise estática e dinâmica de estruturas. Na análise de vibrações método consiste em discretizar a estrutura em «elementos», obtendo matrizes locais de massa e rigidez, realizar o *assembly* para obter matrizes globais de massa e rigidez e então resolver o problema de autovalores e autovetores da Eq. (2-13).

$$[M]^{-1}[K]\{X\} - \lambda[I]\{X\} = \{0\} \tag{2-13}$$

As matrizes locais para o elemento de viga usado neste estudo são:

$$[K_e] = \frac{EI}{L_e^3}\begin{bmatrix} 12 & 6L_e & -12 & 6L_e \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 \\ -12 & -6L_e & 12 & -6L_e \\ 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix} \tag{2-14}$$

$$[M_e] = \frac{\rho A L_e}{420}\begin{bmatrix} 156 & 22L_e & 54 & -13L_e \\ 22L_e & 4L_e^2 & 13L_e & -3L_e^2 \\ 54 & 13L_e & 156 & -22L_e \\ -13L_e & -3L_e^2 & -22L_e & 4L_e^2 \end{bmatrix} \tag{2-15}$$

onde K_e é matriz de rigidez local, M_e é a matriz de massa do elemento finito E é o módulo de Young, I é a inércia de seção transversal, ρ é a massa específica do material da viga, A é a área de seção transversal, L_e é o comprimento do elemento finito.

No contexto da análise de vibrações os autovetores são as frequências naturais e os autovetores os modos de vibração.

2.4 Trabalhos relacionados

O campo de monitoramento estrutural tem evoluído significativamente nas últimas décadas, permitindo a detecção e avaliação de danos em estruturas de forma mais precisa. Dentre as técnicas disponíveis, a análise de vibrações tem se mostrado uma alternativa viável e não invasiva para identificar características de danos a partir de medições dinâmicas. Nesse contexto, diversas abordagens têm sido propostas para a identificação de danos em estruturas e neste texto serão apresentadas algumas delas.

A literatura apresenta diversas abordagens para a modelagem de danos em vigas, sendo que três categorias principais foram identificadas: redução local de rigidez, modelos de mola discretos e modelos complexos em duas ou três dimensões. O trabalho de Friswell e Penny, (FRISWELL; PENNY, 2002), avaliou e comparou essas abordagens, concluindo que para monitoramento da saúde estrutural usando vibrações de baixa frequência, modelos simples, e com poucos graus de liberdade, de elementos de viga são bem adequados. O autor ainda enfatizou que o modelo de redução local de rigidez é uma das abordagens mais comuns para a modelagem de danos em estruturas de viga.

Em 2011, Paulo Lourenço e Luís Ramos (LOURENÇO, 2011) propuseram uma abordagem de identificação de danos em estruturas de alvenaria baseada em medições de vibrações e uma seleção de métodos disponíveis na li-

teratura. Os métodos foram combinados de maneira qualitativa com o objetivo de capturar fenômenos difíceis e auxiliar na decisão da análise de identificação de danos. A metodologia pode ser facilmente aplicada a estruturas de alvenaria e fornece informações sobre a detecção (Nível 1), a localização (Nível 2) e, possivelmente, a avaliação (Nível 3) de danos. Além disso, eles apresentaram uma metodologia baseada em análise modal operacional para estruturas de alvenaria, visando detectar danos em estágios iniciais. A metodologia compreende quatro fases e foi aplicada a dois monumentos complexos portugueses usando sensores convencionais. A frequência de monitoramento parece ser uma medida confiável para detecção de danos.

Gonen e Erduran (GONEN, 2022), por sua vez, propuseram um método híbrido de monitoramento estrutural baseado na fusão de dados de sensores convencionais de aceleração e visão computacional para detecção e localização de danos em pontes. O método foi testado em um estudo de caso envolvendo uma ponte simplesmente apoiada de 50 m de comprimento excitada por uma passagem de trem e vibrações usais de operação. Os resultados indicaram que a utilização de câmeras junto com sensores tradicionais forneceu uma alternativa atraente para aplicações de monitoramento de saúde estrutural baseadas em vibrações. A abordagem híbrida foi capaz de detectar e localizar claramente os danos, mesmo com altos níveis de ruído de medição, e se mostrou muito mais eficaz do que o método de monitoramento convencional com sensores esparsos.

Em um experimento de campo, Kai-Chun Chang e Chul-Woo Kim, (CHANG, 2016), avaliaram a detecção de danos em uma ponte de treliça de aço simplesmente suportada com a aplicação de quatro cenários de danos artificialmente inseridos. As frequências modais e os modos de vibração da ponte foram identificados com alta precisão e acurácia usando análise multivariada autorregressiva auxiliada por um diagrama de estabilização. A detecção de danos baseada em vibrações mostrou-se viável quando a característica sensível a danos foi selecionada adequadamente. Além disso, as medidas de critério de asseguração modal (MAC) e critério de asseguração modal coordenado (COMAC) foram eficazes para detectar os cenários de danos examinados. No entanto, nenhum valor de frequência ou MAC individual foi tão eficaz quanto a combinação de medidas, e nenhum dos valores de taxa de amortecimento individual ou combinado foi sensível ao dano simétrico.

3
Inteligência Artificial

A definição clássica de Inteligência Artificial diz que esta é similar à inteligência humana porém exibida por mecanismos ou softwares. Outra definição mais recente é que ela é um sistema capaz de receber dados, aprender a partir deles e então, após este treinamento, ser capaz de atingir objetivos e tarefas específicas, até mesmo para situações inéditas.
O desenvolvimento tecnológico viabilizou a popularização de técnicas de IA devido a maior velocidade e capacidade de processamento para grandes volumes de dados. Desta forma, é natural inferir que desenvolver Inteligência Artificial com um banco de dados pequeno não é adequado.
Uma "inteligência artificial"normalmente é desenvolvida para tratar de um problema específico. Essas técnicas não são e não devem ser tratadas como uma "caixa preta", existe um conteúdo matemático envolvido nas técnicas que as tornam aplicáveis ou não para cada tipo de problema. Portanto, é necessário investigar e validar a aplicabilidade da técnica de inteligência artificial que se deseja usar antes de tratá-la como solução do seu problema. Não seria adequado, por exemplo, utilizar inteligência artificial para problemas muito simples, a inteligência artificial tem como principal vantagem a possibilidade de atacar problemas complexos.
O macro-grupo "Inteligência Artificial"é popularmente divido em cinco categorias, são elas: aprendizado de máquinas, sistemas especialistas, processamento de linguagem natural, planejamento automatizado e robótica. O escopo deste trabalho serão técnicas de aprendizado de máquina.

3.0.1
Aprendizado de máquinas

O *Machine Learning* corresponde ao desenvolvimento de sistemas por meio de dados ao invés de programação explícita, buscando padrões ou mesmo anomalias escondidas dentro da complexidade do problema. Pode ser categorizado em quatro sub-grupos, são eles: aprendizado supervisionado, não-supervisionado, por reforço e sistemas de recomendação. Este trabalho abordará tecnicás de aprendizado de máquinas supervisionado, das quais destacam-se:

(a) Regressão linear

(b) Regressão logística

(c) Árvores de decisão - *Decision Trees*

(d) Florestas aleatórias - *Random Forests*

(e) *Support Vector Machines*

(f) Redes neurais artificiais

(g) *K-Nearest Neighbors*

3.1 Regressão linear

É um dos métodos supervisionados mais simples. Pode ser aplicado para predizer um *ouput* a partir de um *input* após o treinamento. A ideia é aproximar o fenômeno estudado como um polinômio de 1º grau a múltiplas variáveis e inferir o comportamento y dadas as entradas x_i:

$$h_\theta(X) = \theta_0 + \sum \theta_i * x_i \tag{3-1}$$

Inicialmente são atribuídos valores iniciais para o vetor θ e dentro do grupo de treinamento, onde os valores de X e seus respectivos resultados y são conhecidos, é calculado uma "função de custo"para avaliar a proximidade da predição para esses valores de θ. O objetivo é variar θ para minimizar a "função de custo", que é dada por:

$$J(\theta_i) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{3-2}$$

Onde m é o numero de variáveis.
Desta forma, o problema de minimização tem muitas formas de ser resolvido, a mais popular aplicada em problemas de regressão linear é o chamado *Gradient Descent*. Seu algoritmo pode ser resumido assim:

repetir até convergir

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) \tag{3-3}$$

Que para múltiplas variáveis temos:

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i-1}^{m}(h_\theta(x^{(i)} - y^{(i)})x_j^{(i)} \tag{3-4}$$

Sendo α chamado de taxa de aprendizagem.

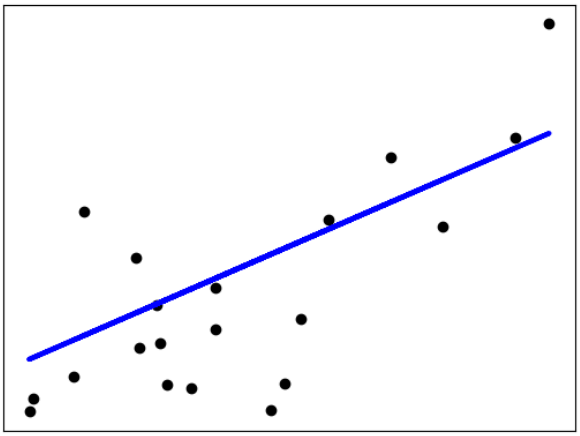

Figura 3.1: Exemplo de regressão linear.

3.2 Regressão logística

A regressão logística é utilizada para problemas de classificação, como por exemplo: o email é spam ou não? Um elemento estrutural possui dano ou não? A ideia central é parecida com o método de regressão linear, porém, desta vez precisamos de uma função hipótese que esteja no limite $0 \leq h_\theta(X) \leq 1$ e desta forma o objetivo se torna definir um limite que separe o que se deseja classificar.

A função hipótese mais usada problemas de regressão logística é:

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} \tag{3-5}$$

A resposta do modelo será um valor entre 0 e 1 e isto representa a probabilidade da classificação dados as váriaveis de entrada. A "função de custo"é dada por:

$$J(\theta_i) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] \tag{3-6}$$

O algoritmo de minimização da "função de custo"é idêntico ao *Gradient Descent* apresentado para Regressão Linear atualizado das funções hipótese e custo.

3.3 Árvore de decisão

Árvores de decisão por vezes são consideradas os melhores algoritmos de aprendizagem de máquina. A ideia central deste algoritmo de aprendizagem

supervisionada gira em torno de uma variável "alvo"predeterminada. O objetivo é separar subconjuntos de dados com características comuns.
A árvore de decisão pode ser comparada a uma série de regras "se ..., então ...". Durante o treinamento o banco de dados é sub-dividido conforme a profundidade da árvore aumenta.

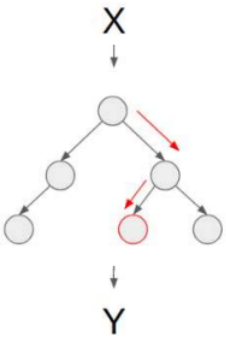

Figura 3.2: Exemplo de árvore de decisão (*decision tree*).

Em termos matemáticos

$$Entropia(S) = -\sum_{c \in C} p(c) log_2(p(c)) \tag{3-7}$$

onde S é o banco de dados do qual se calcula a entropia, c é uma classe de S, $p(c)$ é a proporção de c em S. O valor de entropia varia entre 0 e 1, o que se busca inicialmente são os melhores atributos para separar do banco de dados e alcançar uma árvode de decisões otimizada e, portanto, busca-se a menor entropia. Ganho de informação (GI) é a diferença de entropia antes e depois da separação do banco de dados. O atributo que obtém maior GI implica na melhor separação de dados e classifica melhor a variável "alvo". GI é representado por

$$GI(S, a) = Entropia(S) - \sum_{v \in vcalues(a)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{3-8}$$

onde a é o atributo, $Entropia(S)$ é entropia antes da separação do banco de dados, $|S_v|/|S|$ é a proporção dos valores em após a separação e antes da separação, $Entropia(S_v)$ é a entropia após a separação.

3.4 Florestas aleatórias

Florestas aleatórias é um algoritmo de aprendizagem supervisionada que é baseado no último agoritmo apresentado as árvores de decisão (*Decision Trees* - DT). Basicamente consiste em criar um conjunto de DT independentes e aleatoriamente diferentes entre si para gerar um *output* através da combinação

dos resultados individuais. Cada árvore é treinada com uma subamostragem diferente dentro dos dados de treinamento e a ramificação dos nós de cada árvore usa uma escolha de *features* (características) aleatória. Essas são as duas formas de aleatoriedade introduzidas no método RF que viabilizam uma redução significativa na variância. Por fim, o *output* é extraído em forma da média das probabilidade de classificação das DT.

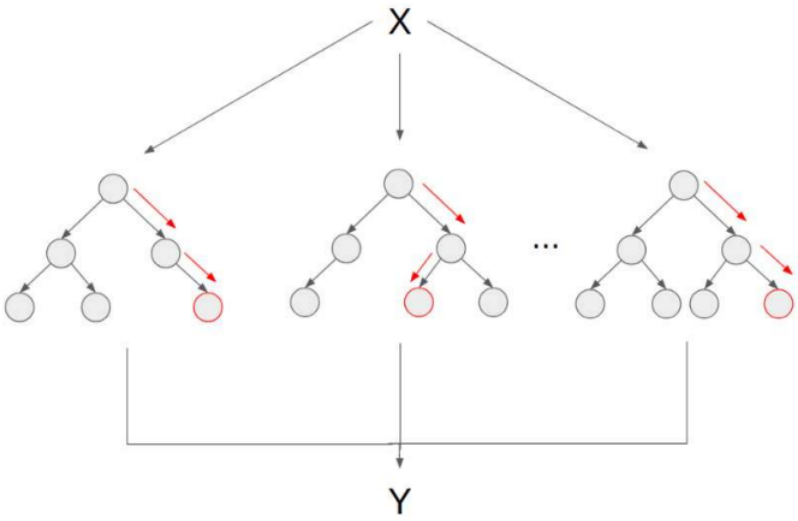

Figura 3.3: Exemplo de floresta aleatória (*random forest*).

3.5 *Support Vector Machines*

Este algoritmo apesar de também ser usado para problemas de regressão é mais indicado para problemas de classificação. O objetivo do *Support Vector Machines* (SVM) é encontrar um hiperplano no espaço dimensional dos exemplos de treinamento que maximize a distância, ou aumentar as margens, entre as classes que se deseje prever.

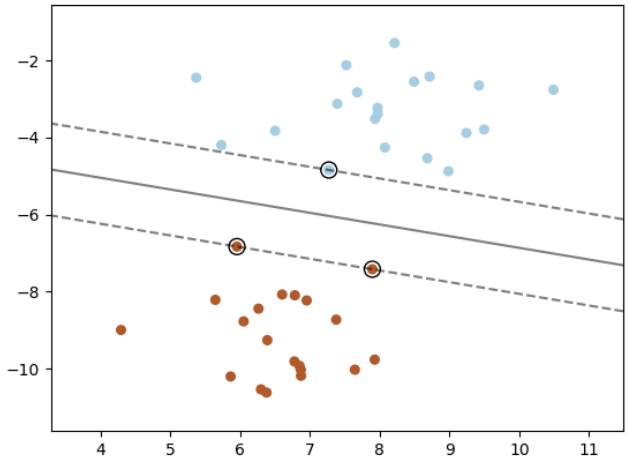

Figura 3.4: Exemplo de maximização de distâncias entre duas classes (*Scikit learn documentation on SVM*).

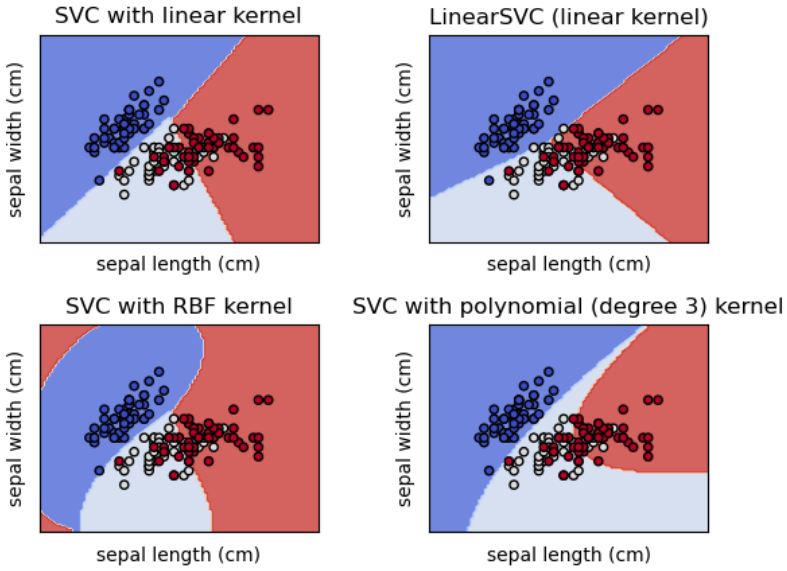

Figura 3.5: Exemplo de diferentes *kernels* na classificação de três classes (*Scikit learn documentation on SVM*).

3.6 Redes neurais artificiais

A arquitetura das redes neurais artificiais (RNA) foi inspirada em células fundamentais presentes no cérebro humano: os neurônios. Os sistemas das RNA são compostos, tipicamente, de três camadas de processamento: entrada, camadas ocultas e saída. Essas camadas são constituídas por unidades básicas de processamento, os neurônios, que são conectados a todos os neurônios

da camada subsequente através de pesos. Esses pesos são, normalmente, multiplicados pelos valores recebidos no neurônio e acrescidos de uma constante (bias). Quando se fala de treino supervisionado temos um conjunto de dados com a resposta desejada, "Target", e esses pesos são modificados através de algoritmos modificados para buscar uma resposta ótima em relação ao "Target". Na Figura 3.6 temos um exemplo ilustrativo de arquitetura de uma rede neural artificial "Multilayer Perceptron".

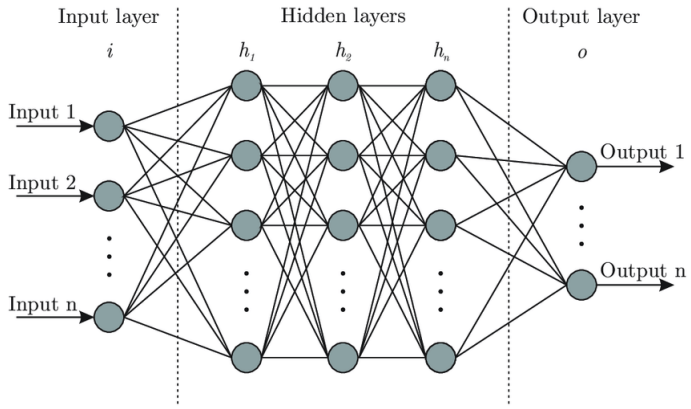

Figura 3.6: Arquitetura genéria de uma rede neural artificial.

A função de ativação muito usada nas redes neurais artificiais é a sigmoidal:

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} \tag{3-9}$$

A resposta do modelo será um valor entre 0 e 1 e isto representa a probabilidade da classificação dados as váriaveis de entrada. A "função de custo"é dada por:

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) log(1-(h_\Theta(x^{(i)}))_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{ji}^{(l)})^2 \tag{3-10}$$

3.7 *K-nearest neighbors*

Este algoritmo classifica um exemplo desconhecido do banco de dados de acordo com os K exemplos, do grupo de treinamento, mais próximos ("k vizinhos mais próximos"). Para isso, inicialmente são calculadas distâncias e, dentre as diferentes maneiras de cálculo, este trabalho usou a distância Euclidiana, dada pela 3-11.

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \quad (3\text{-}11)$$

Uma vez calculadas as distâncias, classifica-se o exemplo de acordo com o grupo em maioria. Este algoritmo possui apenas um parâmetro livre a ser controlado e que se busca otimizar, o número de vizinhos K. É importante observar que no caso de apenas 2 "Targets"(com e sem dano) é recomendado não utilizar um número par de vizinhos.

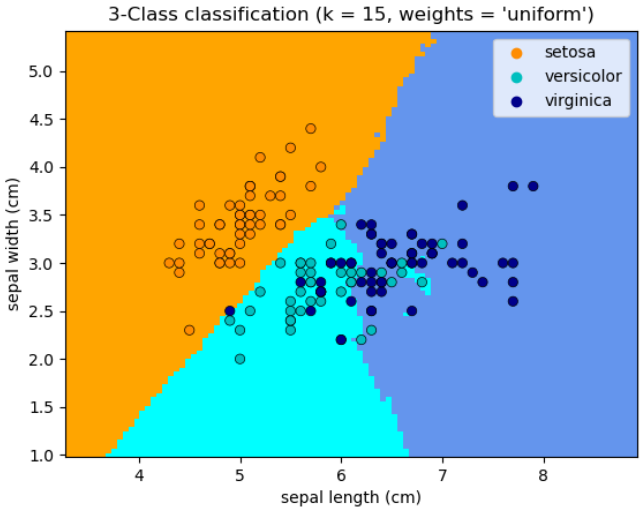

Figura 3.7: Exemplo de *K-nearest neighbor* (*Scikit learn documentation on KNN*).

3.8 Bancos de dados

Bancos de dados são essenciais no contexto de aprendizado de máquinas, pois eles fornecem os dados necessários para que os algoritmos de aprendizagem possam ser treinados e melhorados. Sem um banco de dados robusto e bem estruturado, os algoritmos de *machine learning* não terão informações suficientes para aprender com precisão e produzir resultados relevantes. Portanto, um banco de dados adequado é uma parte crucial de qualquer projeto de aprendizado de máquinas.

Além disso, bancos de dados bem projetados e estruturados podem ajudar a reduzir o tempo e o custo associados à coleta e preparação de dados para uso dos algoritmos. Por fim, não só a qualidade, mas também a quantidade dos dados interferem diretamente no sucesso de qualquer modelo que de aprendizado de máquina que se deseje projetar.

3.9 Métricas de análise

Outrossim, para alcançar o objetivo deste trabalho, que é a comparação de desempenho entre os diferentes métodos de *machine learning* no problema em escopo, é preciso definir métricas objetivas a serem utilizadas de maneira comparativa. Existem algumas métricas estabelecidas no contexto de aprendizado de máquinas para avaliação de métodos e que serão utilizadas neste trabalho no contexto de classificadores.

3.9.1 Acurácia

Consiste na razão da quantidade de predições certas e quantidade de exemplos do conjunto de teste, Eq. 3-12.

$$Acc = \frac{(x - u)}{s} \tag{3-12}$$

3.9.2 Precisão

A precisão de um modelo de classificação é definida como a razão entre o número de exemplos classificados como positivos de maneira correta (positivos verdadeiros) e a soma entre os positivos verdadeiros e os falsos positivos.

$$P = \frac{V_p}{V_p + F_p} \tag{3-13}$$

3.9.3 Revocação

A revocação (*recall*) de um modelo de classificação é definida como a razão entre o número positivos verdadeiros e a soma entre os positivos verdadeiros e os falsos negativos.

$$R = \frac{V_p}{V_p + F_n} \tag{3-14}$$

3.9.4 F1 *Score*

Esta métrica pode ser vista como uma média harmônica de precisão e revocação.

$$F1 = 2 * \frac{(P * R)}{(P + R)} \tag{3-15}$$

3.9.5 Validação cruzada

A validação cruzada, ou *cross-validation*, é maneira alternativa de verificar do desempenho de um modelo para evitar *overfitting* aos dados de treinamento ou até mesmo de teste. Ao invés de separar o banco de dados em três grupos (treinamento, validação e teste), optou-se neste trabalho por realizar uma validação cruzada no grupo de treinamento usando um procedimento chamado *k-fold Cross-Validation*, que consiste em separar o grupo de dados em K subgrupos aleatórios e gerar K modelos diferentes treinados com $K-1$ grupos e testados no grupo excluído naquele treinamento, Figura 3.8. Desta forma, é possível obter uma distribuição na métrica de performance que se deseje analisar. O número K de subgrupos usado neste trabalho foi igual a 10.

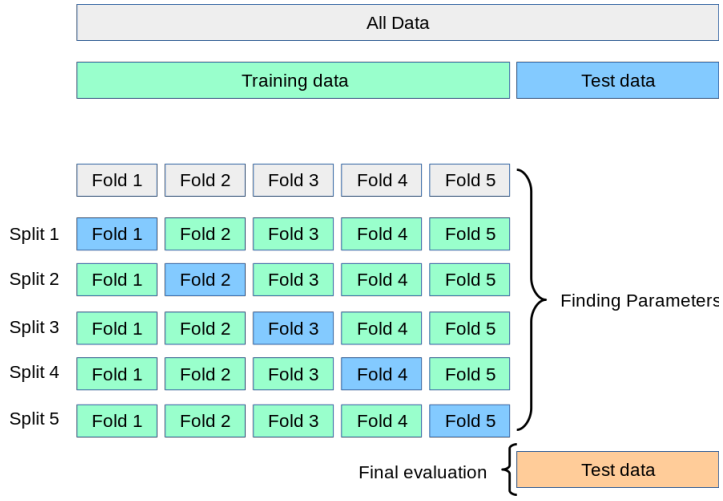

Figura 3.8: Exemplo de *k-fold CV* (*Scikit learn documentation on k-fold CV*).

3.9.6 Matriz de confusão

A matriz de confusão avalia a acurácia do modelo, por definição o elemento ij da matriz representa o número de observações verdadeiras do grupo

i que foram classificadas no grupo j, Figura 3.9. Neste trabalho optou-se pela normalização da matriz, ao invés do número de classificações dos grupos ij, foi calculado o percentual das amostras classificados.

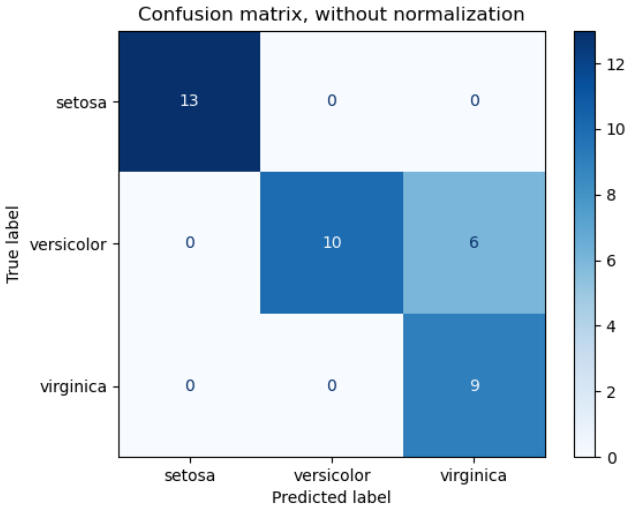

Figura 3.9: Exemplo de Matriz de Confusão (*Scikit learn documentation on Confusion Matrix*).

3.10 Trabalhos relacionados

Os avanços na tecnologia e a disponibilidade de equipamentos de sensoriamento permitiram a coleta contínua de dados de vibração de estruturas. No entanto, processar grandes quantidades de dados de vibração para interpretar danos, especialmente em tempo quase real, é valioso, mas ainda representa um desafio. Modelos de inteligência artificial (IA) provaram ser ferramentas poderosas em SHM baseado em vibração, o que resultou em várias abordagens baseadas em aprendizado de máquina para o diagnóstico de danos em diferentes estruturas. Neste contexto, este trabalho apresenta algumas abordagens da utilização do aprendizado de máquina no contexto de detecção de danos e técnicas baseadas em análise vibracional.

Segundo Sajedi e Liang, (SAJEDI, 2022), o monitoramento da saúde estrutural de pontes é essencial para sua operação segura como parte integrante da infraestrutura de transporte. Com os avanços na tecnologia e a disponibilidade de equipamentos de sensoriamento, a coleta contínua de dados de vibração de pontes se tornou mais acessível. No entanto, processar grandes quantidades de dados de vibração para interpretar danos, especialmente em tempo quase real, é valioso para os interessados, mas ainda representa um desafio devido à escassez de recursos. Modelos de inteligência artificial (IA) provaram ser ferra-

mentas poderosas em SHM baseado em vibração. Este artigo propõe o Trident, uma arquitetura de aprendizado profundo que permite o diagnóstico de danos de alta resolução de estruturas de ponte processando tensores de entrada de vibração 6D. O Trident é validado em um estudo de caso realizado em uma ponte treliçada de aço. O Trident pode servir para outras aplicações de SHM em que matrizes de sensores grandes além de acelerômetros são utilizadas.

O artigo de Sun *et al.* , (SUN LI LIANG, 2018), propõe um método para detecção de danos em pontes baseado em "dynamic fingerprints"e técnicas de aprendizado de máquina, com o intuito de melhorar a efetividade da detecção de danos causados por corrosão ambiental e cargas externas. O método utiliza a análise de vibração para adquirir as "dynamic fingerprints", que são então integradas por meio da fusão Bayesiana para localização preliminar do dano. Além disso, o método combina a teoria do Rough Set e o classificador Naive-Bayes (RSNB) para uma detecção de extensão de dano mais robusta. A validação do método foi feita por meio da comparação com outras técnicas, onde se observou que o RSNB foi superior em termos de transparência, precisão, eficiência, robustez ao ruído e estabilidade. O trabalho destaca a necessidade de refinamento do método para lidar com variações de temperatura e validação em pontes reais.

O trabalho de Kushwah, Sahoo e Joshuva, (KUSHWAH SUDARSAN SAHOO, 2021), apresenta uma abordagem baseada em aprendizado de máquina para a diagnóstico e classificação de falhas em pás de turbinas eólicas. As vibrações nas pás da turbina eólica devido às condições ambientais adversas, ao tamanho enorme, à variação da velocidade do vento e ao funcionamento contínuo podem levar a sérios danos na turbina, resultando na redução de sua produtividade e possível falha futura. Portanto, o monitoramento efetivo da saúde e o diagnóstico de falhas são importantes para evitar danos graves nas turbinas eólicas. Nesse sentido, este estudo teve como objetivo diagnosticar as falhas em uma pá de turbina eólica usando sinais de vibração como sinal medido, adquiridos a partir de um sistema de instrumentação adequado, e classificar as falhas usando diferentes técnicas de aprendizado de máquina. Os resultados mostraram que a técnica de aprendizado de máquina é uma abordagem eficaz para a classificação de falhas em turbinas eólicas, e o sinal de vibração é uma boa escolha como sinal de medição para a detecção e diagnóstico de falhas em pás de turbinas eólicas.

A revisão bibliográfica realizada por Avci *et al.*, (AVCI OSAMA ABDELJABER, 2021), tem como objetivo apresentar uma análise abrangente dos métodos de detecção de danos estruturais baseados em vibração, tanto para aplicações paramétricas quanto não paramétricas, abordando estudos não ba-

seados em *Machine-Learning* e baseados em *Machine-Learning*. A revisão conclui que a maioria dos métodos baseados em *Machine-Learning* envolve duas tarefas principais: extração de características e classificação de características, o que torna esses métodos mais genéricos e vantajosos do que os métodos não baseados em *Machine-Learning* na detecção de danos estruturais baseados em vibração em estruturas civis. Ainda, é recomendado evitar o uso de propriedades modais como características sensíveis ao dano em abordagens de detecção global baseadas em *Machine-Learning*, pois elas são afetadas por fatores além do dano estrutural. Por fim, a revisão aponta que a utilização não convencional de *Machine-Learning* tem se mostrado promissoras para desenvolver novas técnicas de detecção de danos baseadas em vibração que não exigem extração manual de características.

4
Método proposto

4.1
Obtenção de dados de vibração

O primeiro passo deste estudo foi gerar dados para treinar e validar os modelos de *Machine Learning*. Para isto, foi elaborado um *script* em Python 3, Anexo A, que iterativamente cria modelos de vigas em elementos finitos e extrai as 5 primeiras frequências naturais de vibração e seus respectivos modos de vibração. Os parâmetros das vigas geradas foram variados conforme a Tabela 4.1. As vigas sem danos são aquelas com valor de severidade do dano igual a $0,00\%$ de sua altura, a estes valores de severidade do dano não faz sentido variar o posicionamento do dano pois os dados de vibração seriam repetidos 3 vezes a mais do que o necessários, isto é, o banco de dados estaria poluído com 4 vigas iguais em cada variação dos outros parâmetros, Figura 4.2 e Figura 4.3. Uma maneira popular de pré-dimensionar vigas de concreto armado e seção retangular é usar a altura igual a 10,00% do comprimento, por isto, a altura das vigas foram forçadamente enquadradas entre 7,5% e 12,5%. Por fim, as condições de apoio escolhidas foram aquelas que mais se aproximam do comportamento observado em edifícios de concreto armado, Figura 4.1.

O elemento finito utilizado nas modelagens foi o B22, que é um elemento de viga em 2 dimensões e com interpolações quadráticas. O material das vigas modeladas tinham densidade 2500 kg/m^3, , módulo de elasticidade 30 GPa e coeficiente de Poisson 0,2.

Tabela 4.1: Parâmetros do banco de dados de vigas

Parâmetros	Valor inicial	Valor final	Passo	Nr de casos
Comprimento (m)	4,00	8,00	0,25	17
Altura (% do comp.)	0,075	0,125	0,025	3
Largura (m)	0,25	0,35	0,10	2
Severidade (% alt.)	0,00	0,30	0,10	4
Posição do dano*	0,10	0,40	0,10	4
Apoios	Bi-engastado	Engastado e apoiado	-	2
Total de casos				2.652

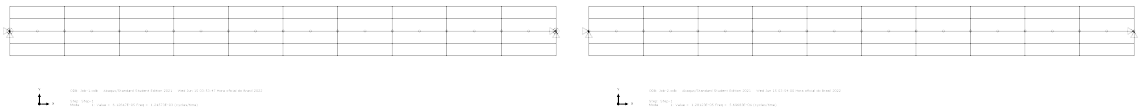

Figura 4.1: Vigas bi-engastada e bi-apoiada sem dano. Os elementos são todos lineares, a seção é apresentada desta maneira apenas de forma ilustrativa para identificação da variação da rigidez em determinadas seções

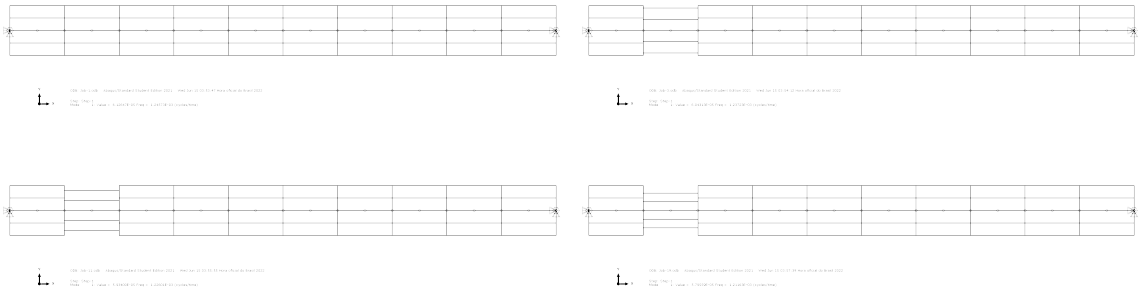

Figura 4.2: Vigas sem dano, dano 10%, 20% e 30% da altura. Os elementos são todos lineares, a seção é apresentada desta maneira apenas de forma ilustrativa para identificação da variação da rigidez em determinadas seções

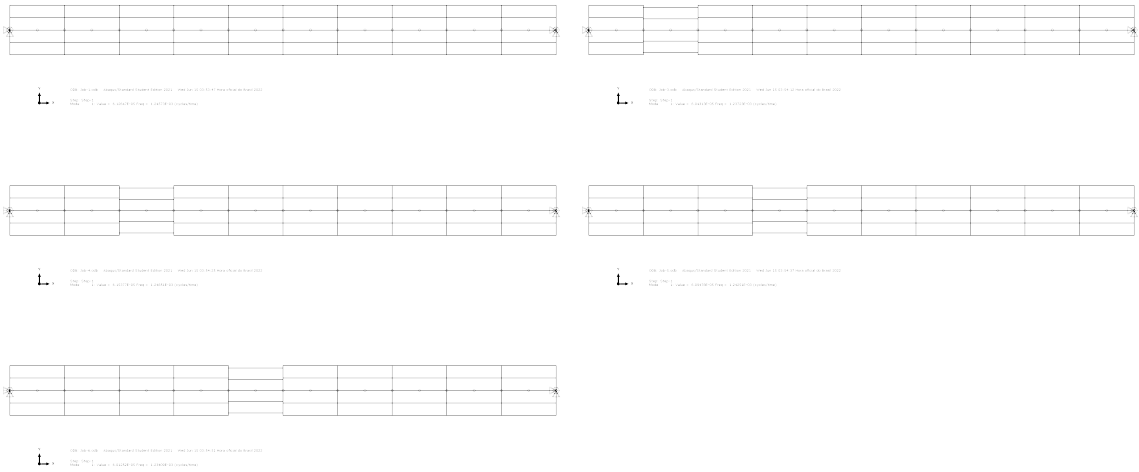

Figura 4.3: Vigas sem dano, dano na posição 1, 2, 3 e 4. Os elementos são todos lineares, a seção é apresentada desta maneira apenas de forma ilustrativa para identificação da variação da rigidez em determinadas seções

4.2
Tratamento de dados

4.2.1 Amostragem

Os dados gerados estão fortemente desbalanceados (Figura 4.4). Após a separação dos dados em treino e teste dois tratamentos foram feitos: *under-sampling* e *over-sampling*. Estes tratamentos nos dados podem ser vistos na Figura 4.4. *Under-sampling* consiste em reduzir aleatoriamente o maior *label* para o mesmo tamanho do menor. De outra maneira, o *over-sampling* cria novos exemplos para o menor *label*, tradicionalmente duplicando dados. Como duplicar dados não parece uma solução viável para o problema, o método escolhido para *over-sampling* foi o *Synthetic Minority Over-sampling Technique (SMOTE)*, Chawla, que cria dados novos a partir de pequenas pertubações e interpolações dos dados existentes. Apesar de serem dados artificiais isto é adequado ao problema considerando que na análise modal experimental existem mais ruídos nos dados de vibração do que nas condições computacionais abordadas neste trabalho. Foram identificados que na geração dos dados algumas vigas estavam com o 4º e o 5º modos de vibração zerados e estes exemplos foram excluídos do banco de dados original, caindo assim de 2.652 exemplos para 2.007.

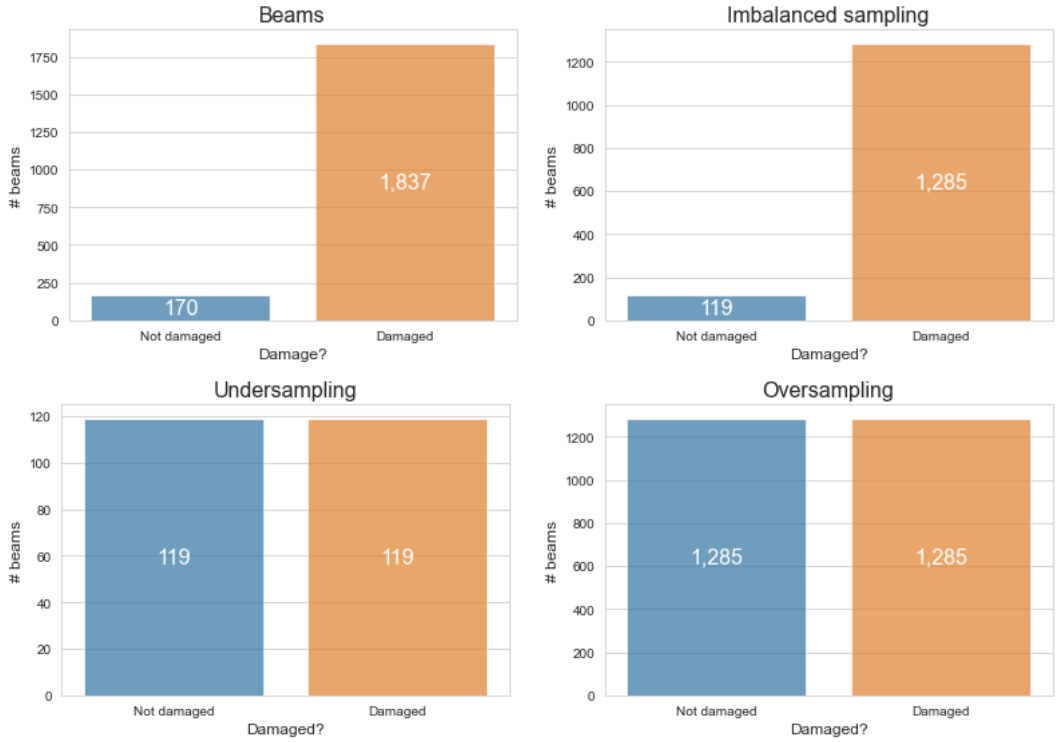

Figura 4.4: Amostras de vigas, grupo de teste *imbalanced*, *under-sample* e *over-sample*

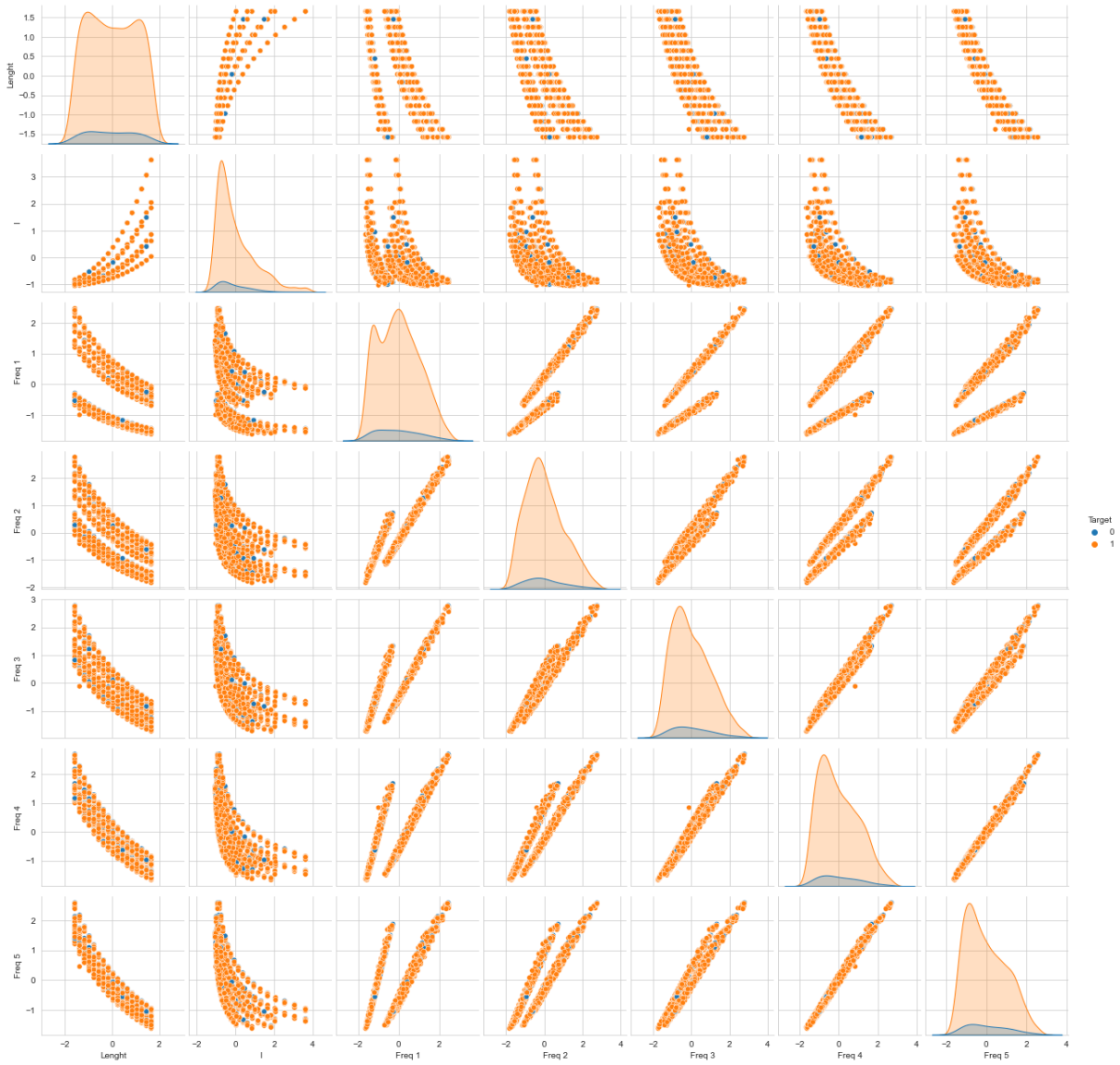

Figura 4.5: Distribuição de exemplos entre pares de características - amostra desbalanceada.

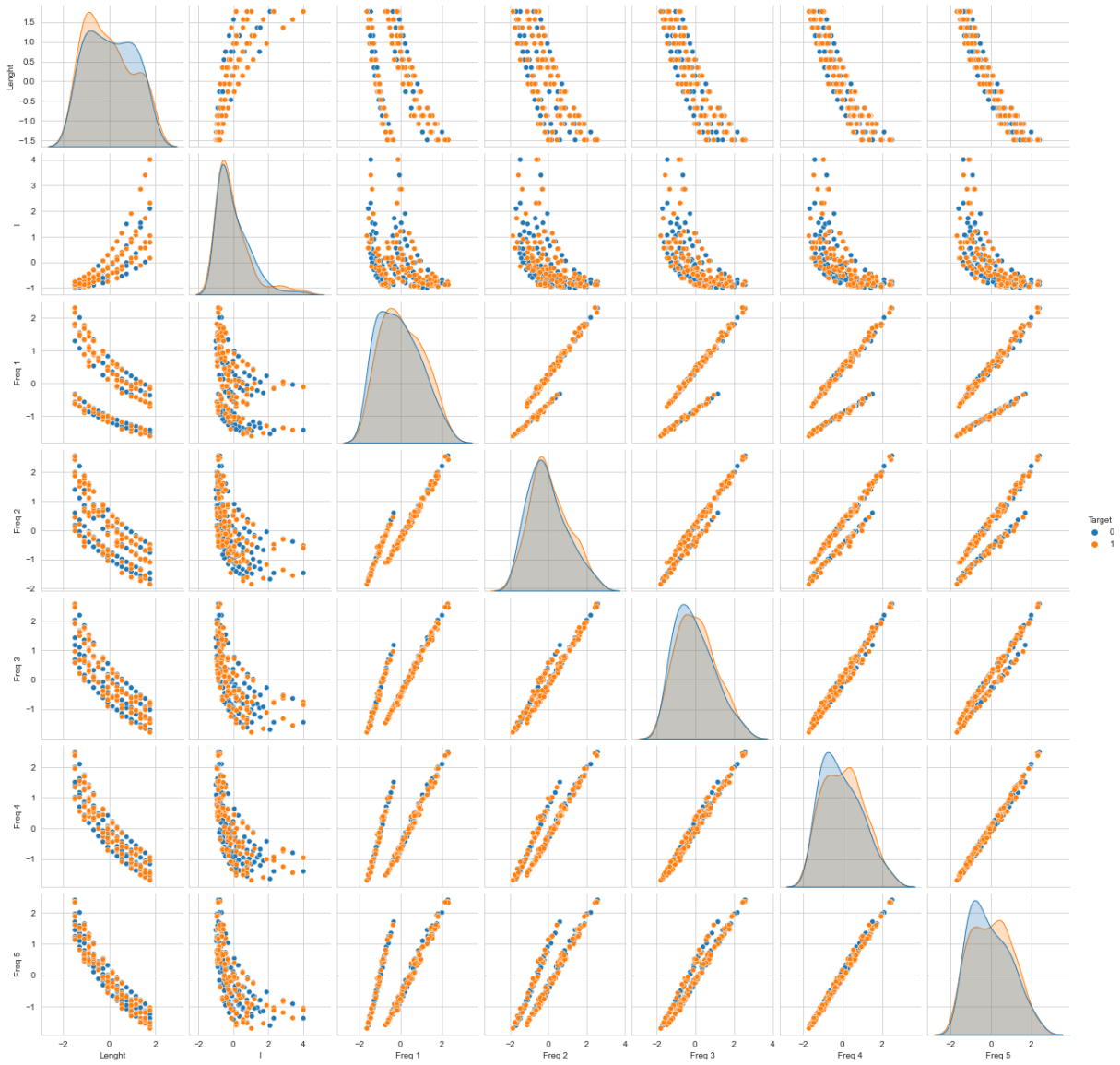

Figura 4.6: Distribuição de exemplos entre pares de características - sub-amostragem.

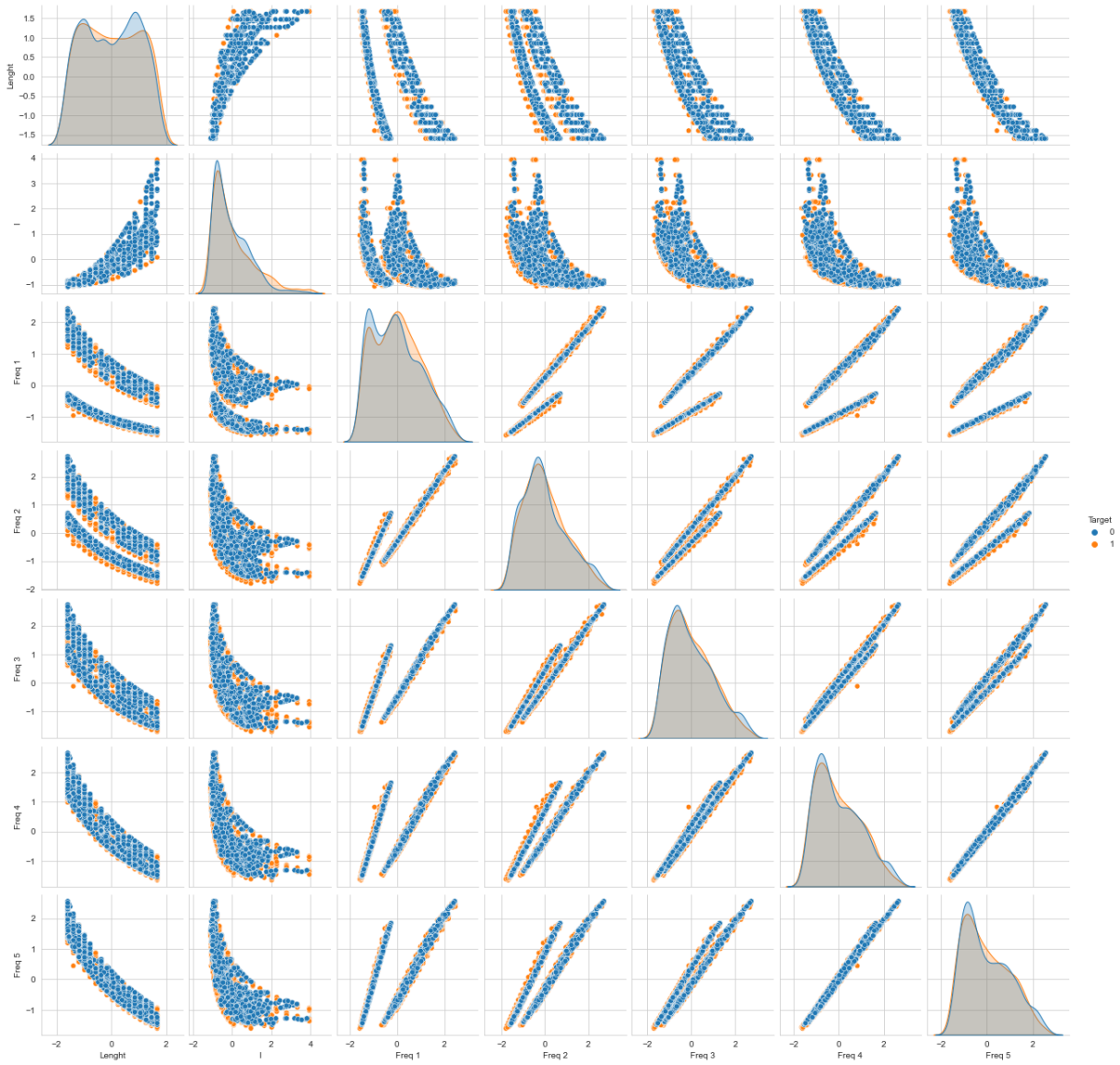

Figura 4.7: Distribuição de exemplos entre pares de características - sobre-amostragem.

4.2.2 *Features scaling*

Features scaling é um passo importante de pré-processamento que busca evitar um viés do algoritmo sobre diferentes entradas com ordem de grandeza diferente. Em outras palavras, informações com valor numéricos muito altos poderão ter mais importância no modelo do que as de valores muito baixos.

Neste estudo foi utilizada a classe de pré-processamento do *Scikit Learn* *sklearn.preprocessing.StandardScaler*, que uniformiza as propriedades dos exemplos através da média e escalando para uma variância unitária. O "*standard score*"de um exemplo x é calculado conforme a equação 4-1,

$$z = \frac{(x - u)}{s} \tag{4-1}$$

onde, u é a média dos exemplos de treino e s é o desvio padrão.

4.3 Modelos

Para cada modelo foi feito um estudo preliminar para identificar a sensibilidade dos hiperparâmetros e, então, viabilizar uma comparação entre modelos cada um dentro da sua melhor performance por tipo de amostragem.

4.3.1 Regressão linear

Para os modelos de classificação baseado em regressão linear, foram exploradas as rotinas: *solver "lsqr"*, baseada no método mínimos quadrados regularizados e *solver "sag"*, baseado no método *Stochastic Average Gradient descent* (SCHMIDT NICOLAS LE ROUX, 2017). A tolerância para convergência das iterações testadas foram: 0.1, 0.01, 0.001 e 0.0001.

4.3.2 Regressão logística

Nos modelos de classificação baseado em regressão logística, foram exploradas os algoritmos de otimização: *solver "lbfgs"*, baseada no método Broyden–Fletcher–Goldfarb–Shanno e *solver "saga"*, que é uma variação baseada no método *Stochastic Average Gradient descent* (SCHMIDT NICOLAS LE ROUX, 2017). A tolerância para convergência das iterações testadas foram: 0.1, 0.01, 0.001 e 0.0001.

4.3.3 *Decision Tree*

Para os modelos baseados em árvores de decisão, buscou-se um desempenho melhor variando a profundidade dos nós (*max depth*): 5, 10, 15, 20, 25, 30, 35 e 40.

4.3.4 *Random Forest*

Para os modelos baseados em florestas aleatórias, buscou-se um desempenho melhor variando a profundidade dos nós (*max depth*): 5, 10, 15, 20, 25, 30, 35 e 40.

4.3.5
Support Vector Machines

As classificações dos métodos de máquinas de vetores de suporte (SVM) utilizaram diferentes funções *kernel*: linear, polinomial, RBF e sigmoidal. E as tolerâncias para convergência: 0.01 e 0.001.

4.3.6
Redes neurais artificiais

Com o algoritmo de redes neurais artificiais buscou-se otimizar o número de neurônios por camada: 5, 10, 15 e 20. E o numéro de camadas internas: 1 e 2.

4.3.7
K-Nearest Neighbor

Com os modelos de K vizinhos mais próximos (KNN) variou-se o numéro K de "vizinhos": 1, 3, 5, 15, 31, 51, 101 e 151.

Algoritmo	Solver	Tolerância	Profundidade	Kernel	Camadas	Neurônios/camada	K Vizinhos
LINr 1	lsqr	0.1	-	-	-	-	-
LINr 2	lsqr	0.01	-	-	-	-	-
LINr 3	lsqr	0.001	-	-	-	-	-
LINr 4	lsqr	0.0001	-	-	-	-	-
LINr 5	sag	0.1	-	-	-	-	-
LINr 6	sag	0.01	-	-	-	-	-
LINr 7	sag	0.001	-	-	-	-	-
LINr 8	sag	0.0001	-	-	-	-	-
LOGr 1	saga	0.1	-	-	-	-	-
LOGr 2	saga	0.01	-	-	-	-	-
LOGr 3	saga	0.001	-	-	-	-	-
LOGr 4	saga	0.0001	-	-	-	-	-
LOGr 5	lbfgs	0.1	-	-	-	-	-
LOGr 6	lbfgs	0.01	-	-	-	-	-
LOGr 7	lbfgs	0.001	-	-	-	-	-
LOGr 8	lbfgs	0.0001	-	-	-	-	-
DT 05	-	-	5	-	-	-	-
DT 10	-	-	10	-	-	-	-
DT 15	-	-	15	-	-	-	-
DT 20	-	-	20	-	-	-	-
DT 25	-	-	25	-	-	-	-
DT 30	-	-	30	-	-	-	-
DT 35	-	-	35	-	-	-	-
DT 40	-	-	40	-	-	-	-
RF 05	-	-	5	-	-	-	-
RF 10	-	-	10	-	-	-	-
RF 15	-	-	15	-	-	-	-
RF 20	-	-	20	-	-	-	-
RF 25	-	-	25	-	-	-	-
RF 30	-	-	30	-	-	-	-
RF 35	-	-	35	-	-	-	-
RF 40	-	-	40	-	-	-	-
SVM 1	-	0.01	-	linear	-	-	-
SVM 2	-	0.001	-	linear	-	-	-
SVM 3	-	0.01	-	poly	-	-	-
SVM 4	-	0.001	-	poly	-	-	-
SVM5	-	0.01	-	rbf	-	-	-
SVM6	-	0.001	-	rbf	-	-	-
SVM 7	-	0.01	-	sigmoid	-	-	-
SVM 8	-	0.001	-	sigmoid	-	-	-
MLP 1	-	-	-	-	1	5	-
MLP 2	-	-	-	-	2	5	-
MLP 3	-	-	-	-	1	10	-
MLP 4	-	-	-	-	2	10	-
MLP 5	-	-	-	-	1	15	-
MLP 6	-	-	-	-	2	15	-
MLP 7	-	-	-	-	1	20	-
MLP 8	-	-	-	-	2	20	-
KNN 1	-	-	-	-	-	-	1
KNN 2	-	-	-	-	-	-	3
KNN 3	-	-	-	-	-	-	5
KNN 4	-	-	-	-	-	-	15
KNN 5	-	-	-	-	-	-	31
KNN 6	-	-	-	-	-	-	51
KNN 7	-	-	-	-	-	-	101
KNN 8	-	-	-	-	-	-	151

Tabela 4.2: Resumo de hiperparâmetros estudados

5
Resultados

Como pode-se observar da Tabela 5.1 e Figuras 5.1 e 5.2 os modelos de classificação baseados em regressão linear classificaram os exemplos como dos grupos de treino, validação-cruzada e teste como vigas danificadas.

Tabela 5.1: Regresão linear - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LINr 01	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 02	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 03	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 04	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 05	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 06	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 07	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LINr 08	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0

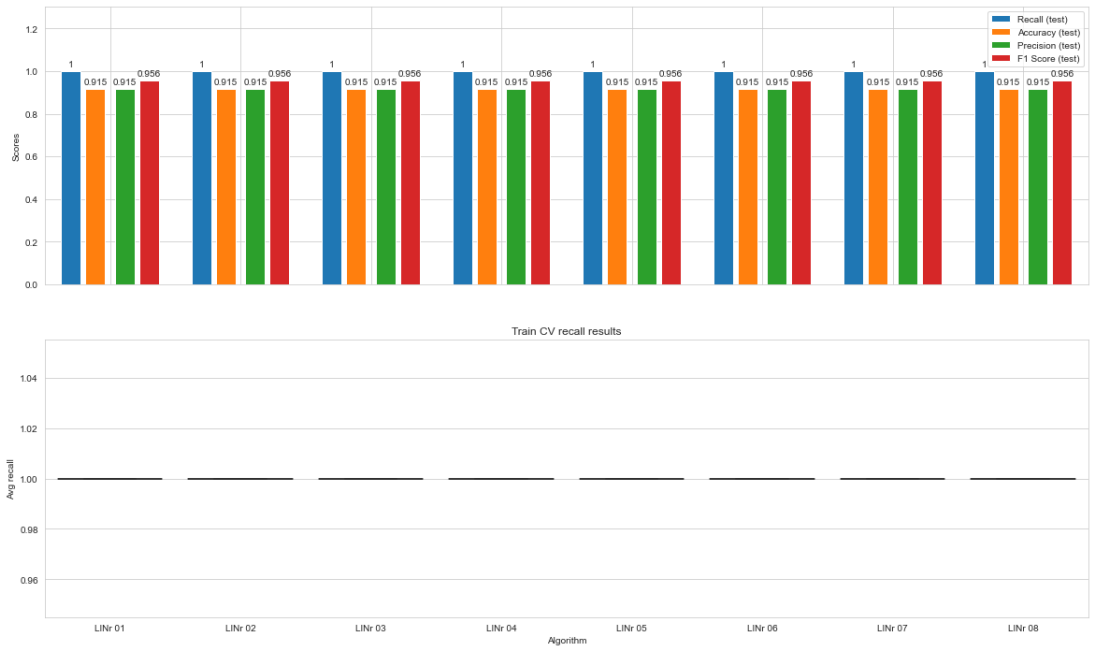

Figura 5.1: Métricas de análise - Regressão linear - amostra desbalanceada

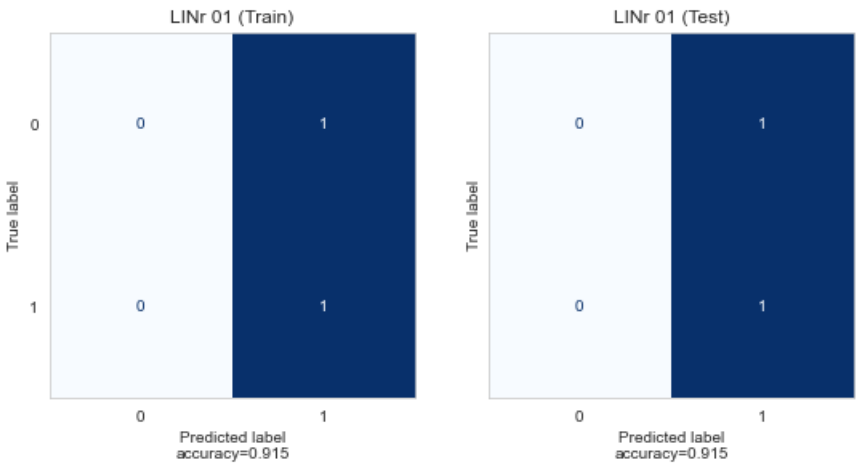

Figura 5.2: Matriz de confusão - Regressão linear - amostra desbalanceada

No caso de sub-amostragem, Tabela 5.2 e Figuras 5.3 e 5.4, os modelos tiveram um desempenho mais confiável, com uma revocação de 54% e uma precisão de 93% no grupo de teste.

Tabela 5.2: Regresão linear - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LINr 02	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.564
LINr 03	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.564
LINr 04	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.564
LINr 06	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.547
LINr 07	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.547
LINr 08	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.564
LINr 05	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.556
LINr 01	0.538	0.438	0.521	0.454	0.520	0.927	0.529	0.595	0.548

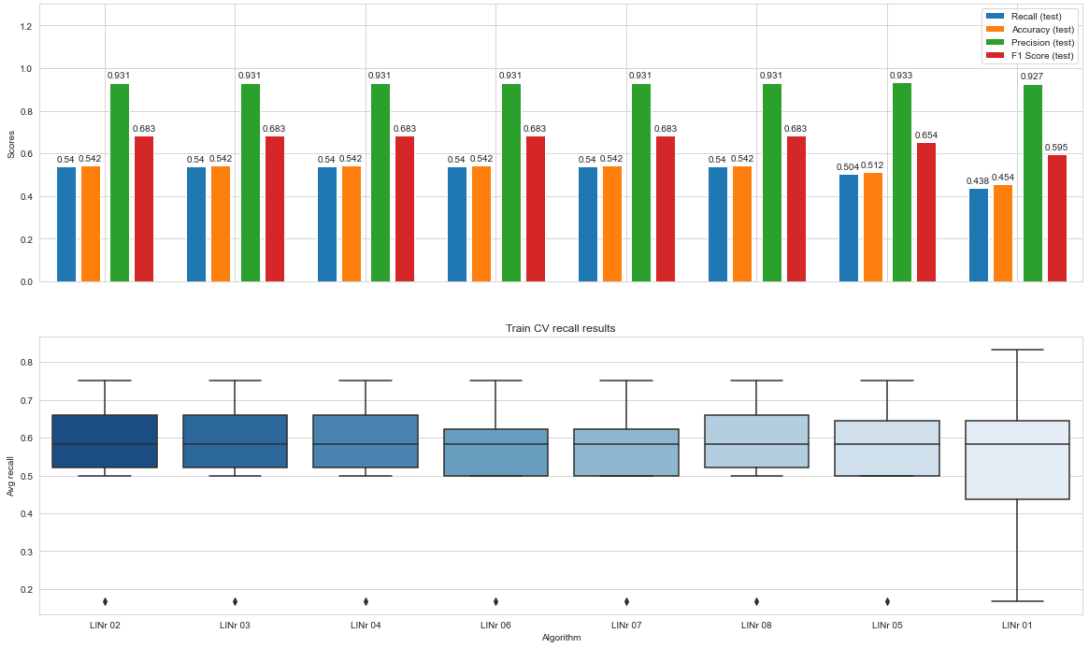

Figura 5.3: Métricas de análise - Regressão linear- sub-amostragem

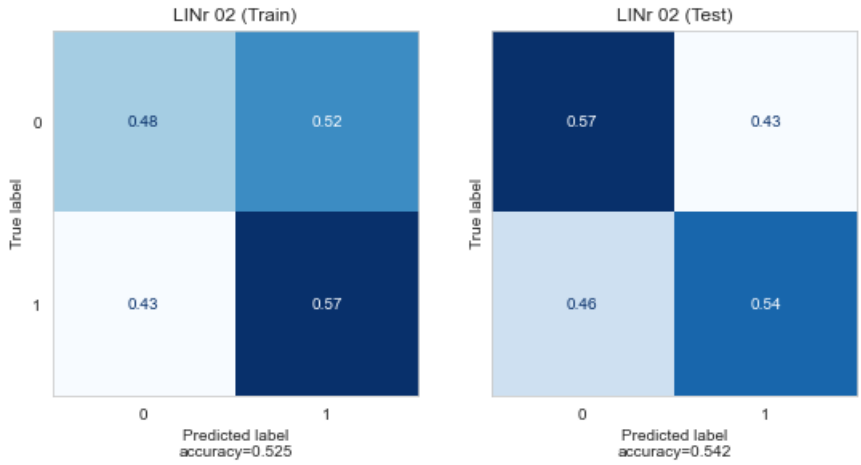

Figura 5.4: Matriz de confusão - Regressão linear - sub-amostragem

O modelo de regressão linear no caso sobre-amostragem, Tabela 5.3 e Figuras 5.5 e 5.6, obteve um desempenho ainda melhor que o caso sub-amostrado, com uma revocação de 66% e uma precisão de 93% no grupo de teste.

Tabela 5.3: Regresão linear - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LINr 02	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 03	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 04	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 05	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.663
LINr 06	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 07	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 08	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LINr 01	0.514	0.565	0.526	0.557	0.527	0.920	0.520	0.700	0.547

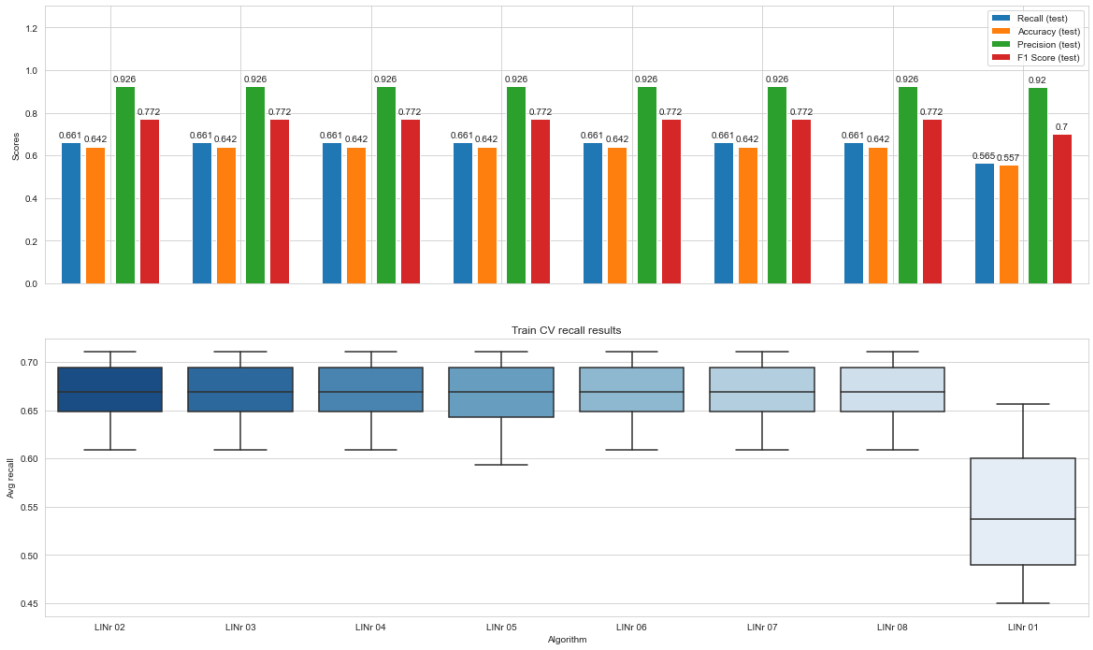

Figura 5.5: Métricas de análise - Regressão linear - sobre-amostragem

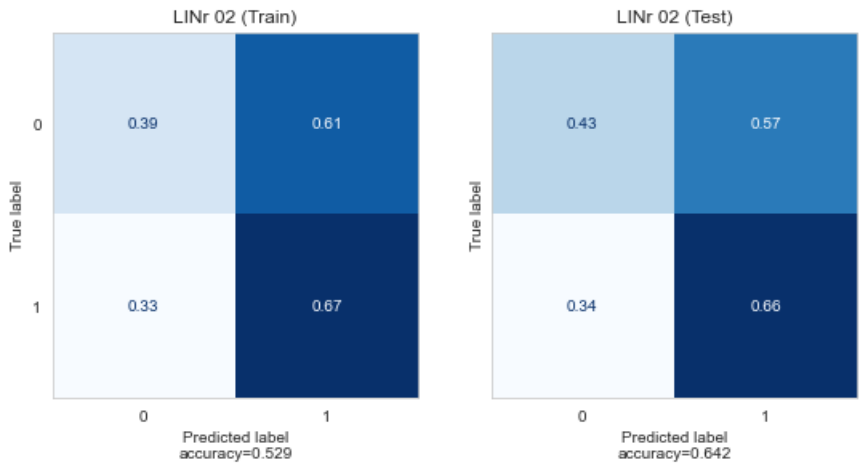

Figura 5.6: Matriz de confusão - Regressão linear - sobre-amostragem

Assim como os modelos de regressão linear, os modelos de regressão logística apresentaram resultados insatisfatórios no caso da amostragem desbalanceada. Os modelos novamente classificaram todas as vigas como danificadas.

Tabela 5.4: Regresão logística - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LOGr 01	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 02	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 03	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 04	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 05	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 06	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 07	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
LOGr 08	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0

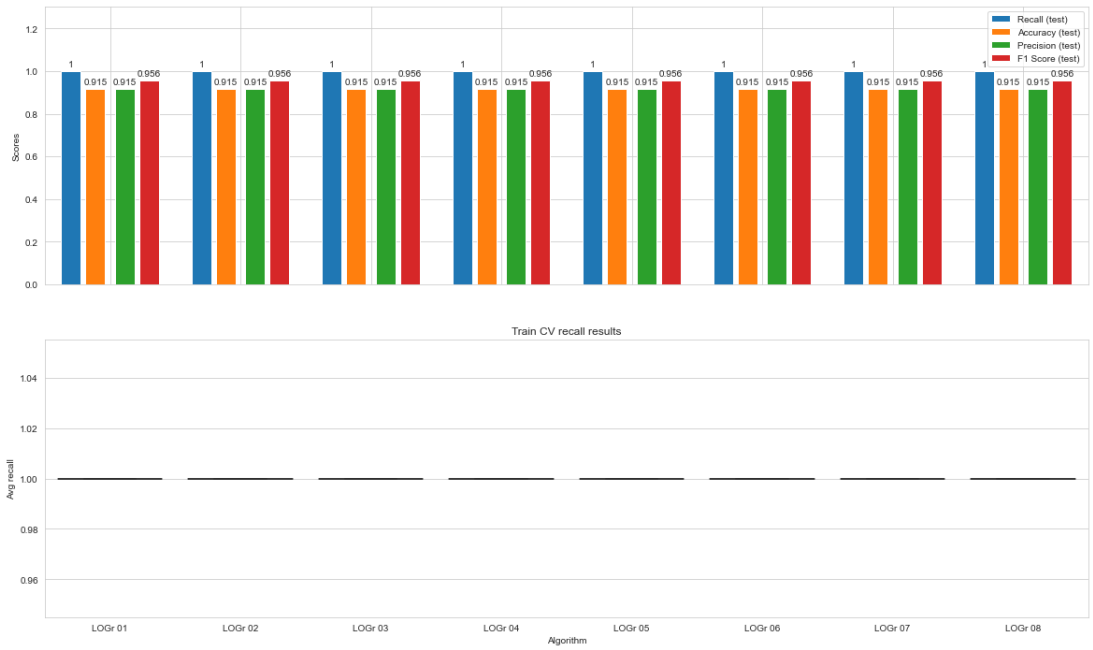

Figura 5.7: Métricas de análise - Regressão logística - amostra desbalanceada

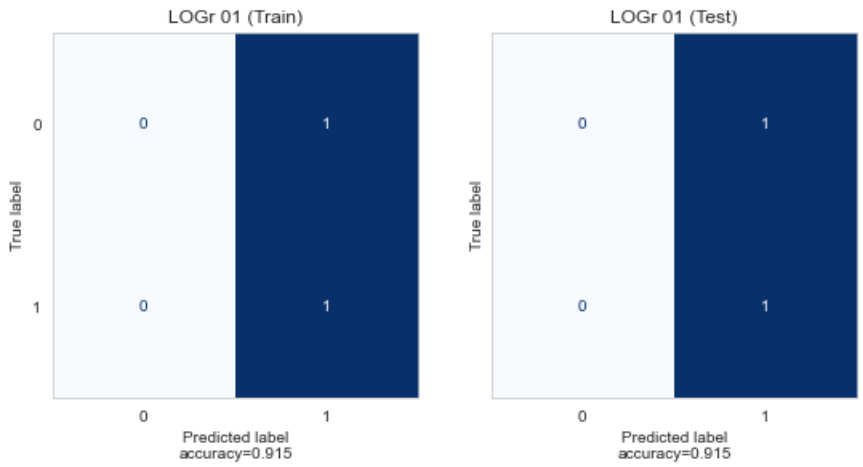

Figura 5.8: Matriz de confusão - Regressão logística - amostra desbalanceada

No caso de sub-amostragem, Tabela 5.5 e Figuras 5.9 e 5.10, os modelos se comportaram de maneira semelhante ao que foi observado nos modelos de regressão linear e o desempenho apresentado é mais confiável que o caso de amostragem anterior. A revocação calculada foi cerca de 57% e a precisão 93% no grupo de teste para o modelo LOGr 01, isto supera o desempenho da melhor performance de regressão linear no caso de amostragem análogo.

Tabela 5.5: Regresão logística - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LOGr 01	0.597	0.569	0.521	0.566	0.518	0.929	0.555	0.706	0.622
LOGr 02	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.547
LOGr 03	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547
LOGr 04	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547
LOGr 05	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547
LOGr 06	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547
LOGr 07	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547
LOGr 08	0.546	0.504	0.529	0.512	0.528	0.933	0.537	0.654	0.547

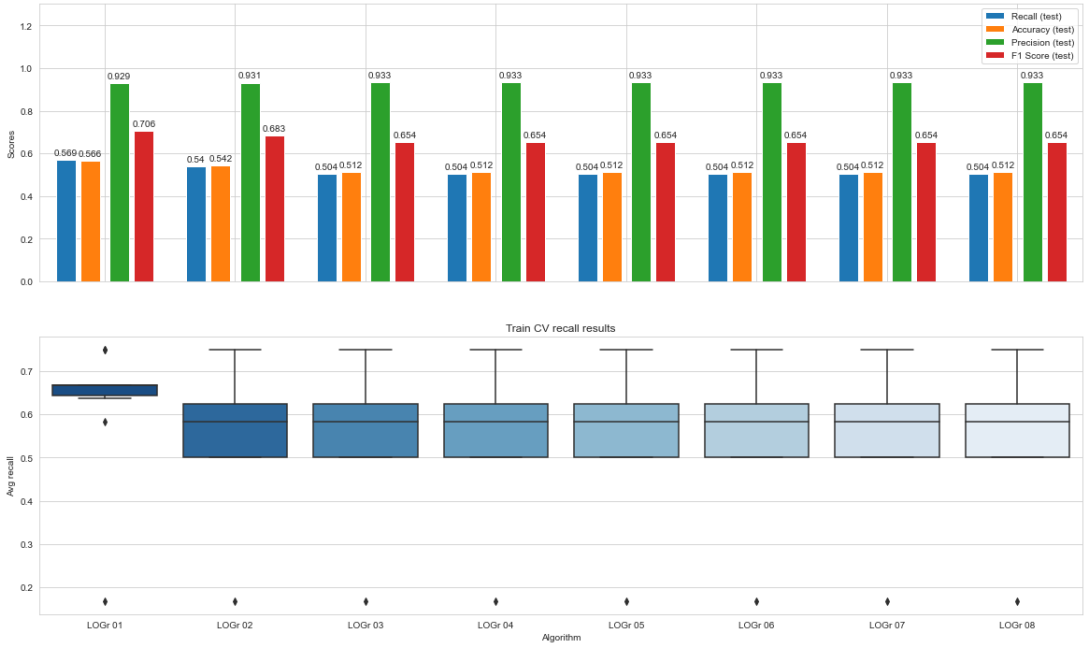

Figura 5.9: Métricas de análise - Regressão logística - sub-amostragem

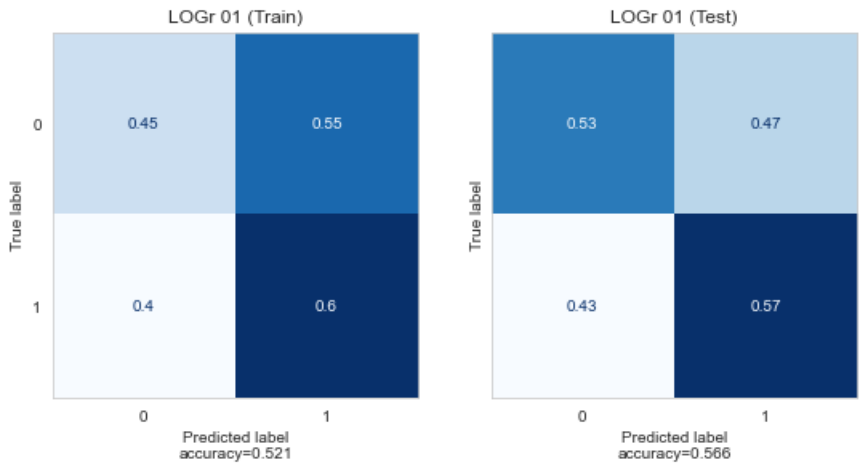

Figura 5.10: Matriz de confusão - Regressão logística - sub-amostragem

Os métodos de regressão logística quando alimentados de uma base sobre-amostrada, Tabela 5.6 e Figuras 5.11 e 5.12, apresentaram nova melhora de desempenho em relação ao caso sub-amostrado. Os resultados foram semelhantes ao caso análogo de regressão linear.

Tabela 5.6: Regresão logística - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LOGr 01	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 02	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 03	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 04	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 05	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 06	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 07	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr 08	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668

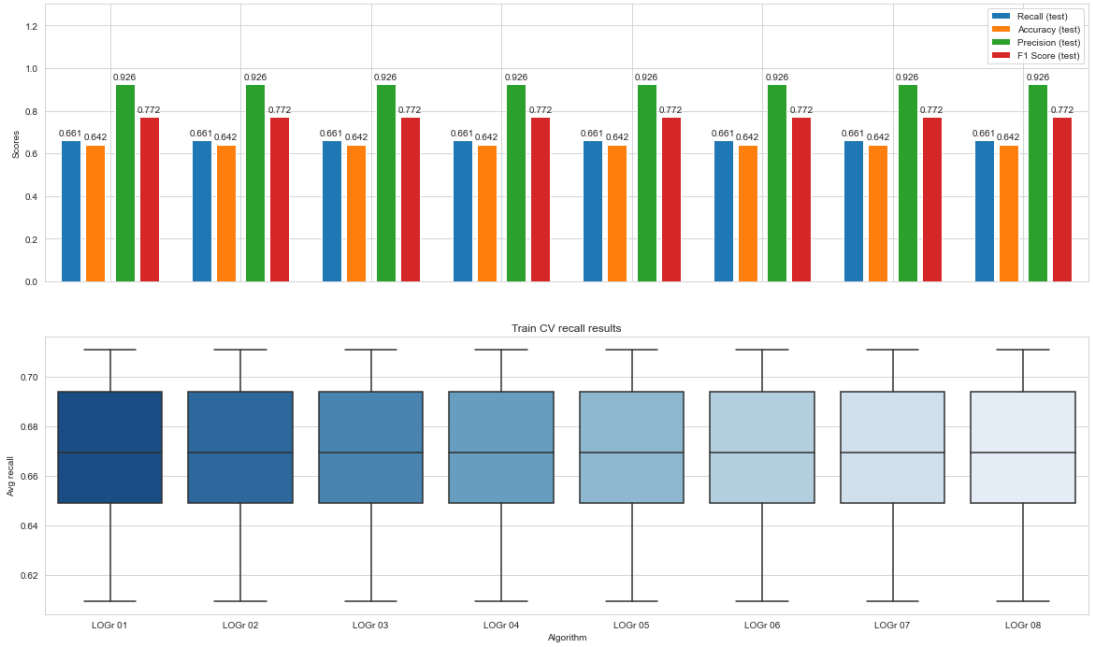

Figura 5.11: Métricas de análise - Regressão logística - sobre-amostragem

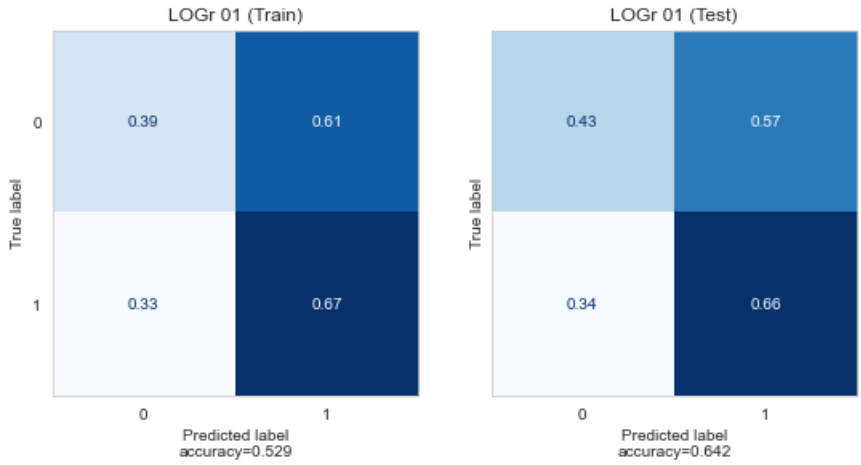

Figura 5.12: Matriz de confusão - Regressão logística - sobre-amostragem

Os modelos de árvore de decisão não se comportaram satisfatoriamente no caso de amostragem desbalanceada. Apesar de *scores* altos em todas as métricas de análise, Tabela 5.7 e Figura 5.13, as matrizes de confusão, Figura 5.14, indicaram um número elevado de classificações "falso positivas".

Tabela 5.7: Árvore de decisão - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
DT 05	1.000	0.996	0.924	0.914	0.923	0.917	0.960	0.955	0.998
DT 10	0.998	0.991	0.939	0.919	0.939	0.926	0.968	0.957	0.992
DT 15	1.000	0.991	0.951	0.927	0.949	0.933	0.974	0.961	0.989
DT 20	1.000	0.989	0.964	0.935	0.962	0.943	0.981	0.966	0.977
DT 25	1.000	0.986	0.970	0.935	0.968	0.946	0.984	0.965	0.969
DT 30	1.000	0.978	0.981	0.932	0.980	0.949	0.990	0.963	0.965
DT 35	1.000	0.978	0.987	0.939	0.986	0.956	0.993	0.967	0.965
DT 40	1.000	0.969	0.994	0.934	0.994	0.959	0.997	0.964	0.962

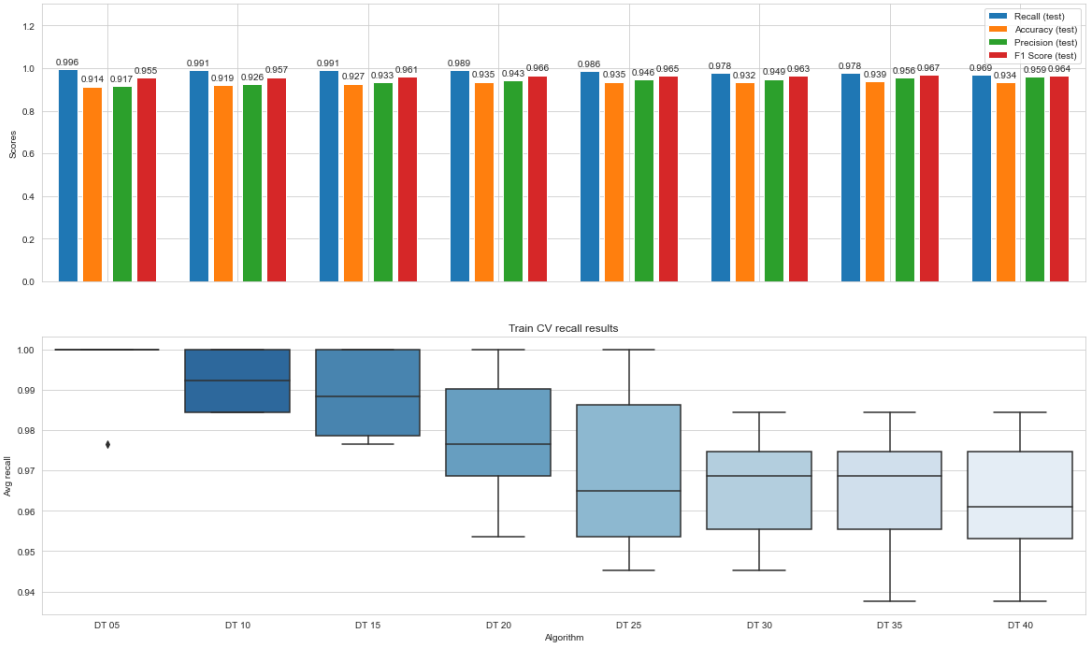

Figura 5.13: Métricas de análise - Árvore de decisão - amostra desbalanceada

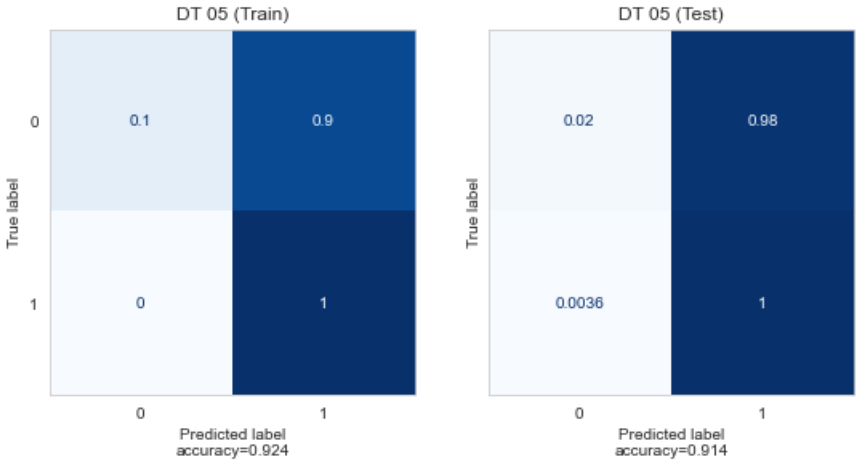

Figura 5.14: Matriz de confusão - Árvore de decisão - amostra desbalanceada

A performance das árvores de decisão para o caso de sub-amostragem foi melhor do quando treinadas com dados desbalanceados, a revocação calculada foi de 80% e uma precisão de 91% nos dados de teste, além de poucas classificações do tipo "falsos negativos", Figuras 5.15 e 5.16. O *score* revocação na validação cruzada ficou esparso.

Tabela 5.8: Árvore de decisão - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
DT 05	0.950	0.808	0.630	0.756	0.579	0.916	0.720	0.859	0.640
DT 10	0.992	0.739	0.735	0.700	0.656	0.917	0.789	0.818	0.538
DT 15	0.992	0.616	0.857	0.599	0.781	0.919	0.874	0.738	0.436
DT 20	1.000	0.525	0.941	0.534	0.895	0.939	0.944	0.674	0.511
DT 30	1.000	0.476	1.000	0.502	1.000	0.960	1.000	0.637	0.486
DT 35	1.000	0.476	1.000	0.502	1.000	0.960	1.000	0.637	0.486
DT 40	1.000	0.476	1.000	0.502	1.000	0.960	1.000	0.637	0.486
DT 25	0.992	0.435	0.987	0.464	0.983	0.956	0.987	0.598	0.478

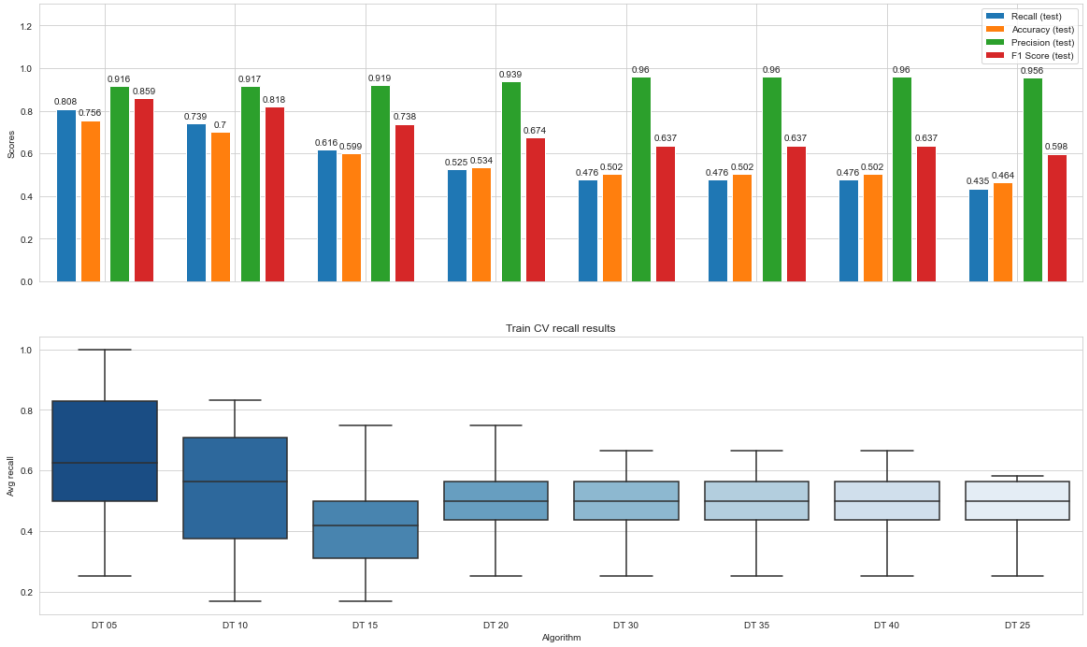

Figura 5.15: Métricas de análise - Árvore de decisão - sub-amostragem

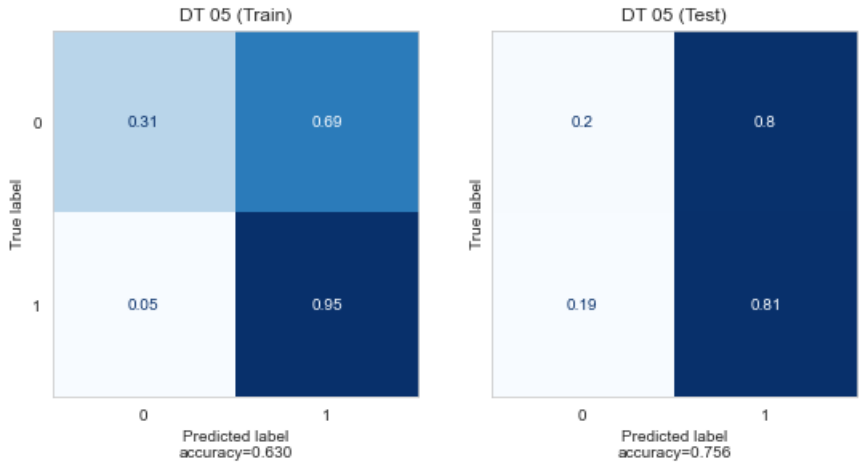

Figura 5.16: Matriz de confusão - Árvore de decisão - sub-amostragem

As árvores de decisão treinadas com dados obtidos da sobre-amostragem se mostraram mais performantes do que os outros casos. *Scores* de revocação obtido igual a 89% e precisão de 93% no grupo de teste, além de boa distribuição de revocações na validação cruzada nos modelos com *depth* maior que 25.

Tabela 5.9: Árvore de decisão - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
DT 35	1.000	0.889	1.000	0.837	1.000	0.930	1.000	0.909	0.873
DT 40	1.000	0.889	1.000	0.837	1.000	0.930	1.000	0.909	0.865
DT 30	1.000	0.886	1.000	0.836	0.999	0.931	1.000	0.908	0.870
DT 25	0.977	0.882	0.981	0.831	0.985	0.929	0.981	0.905	0.828
DT 05	0.840	0.853	0.589	0.793	0.559	0.915	0.671	0.883	0.533
DT 20	0.833	0.741	0.902	0.706	0.966	0.923	0.895	0.822	0.731
DT 10	0.696	0.663	0.691	0.640	0.689	0.922	0.693	0.771	0.594
DT 15	0.616	0.569	0.784	0.556	0.926	0.913	0.740	0.701	0.679

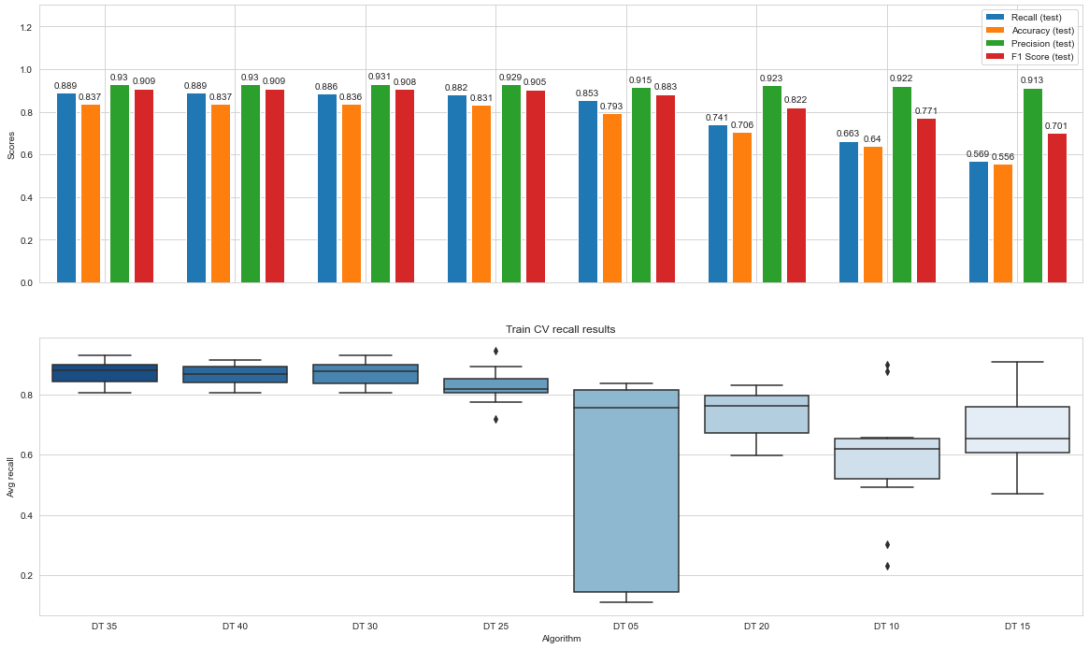

Figura 5.17: Métricas de análise - Árvore de decisão - sobre-amostragem

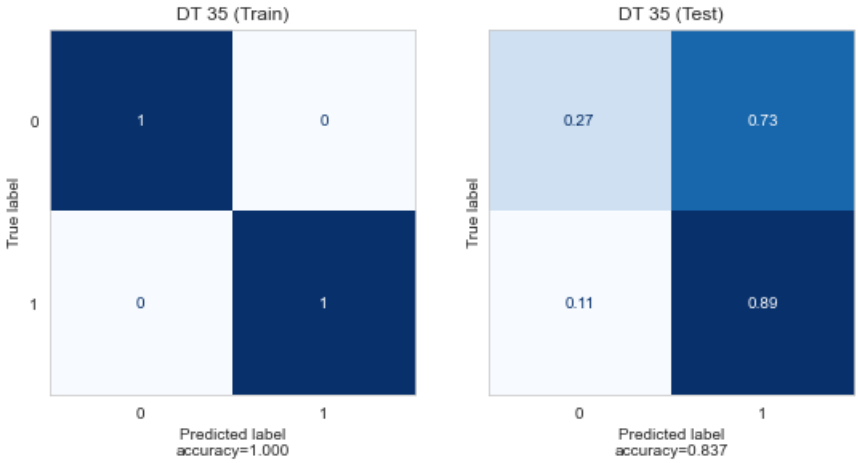

Figura 5.18: Matriz de confusão - Árvore de decisão - sobre-amostragem

O banco de dados desbalanceado quando usados para treinamento de Florestas Aleatórias resultou em uma resposta insatisfatória. O viés do alto número de vigas com dano em relação às sãs implicou em uma classificação próxima de 100% das vigas como "danificadas".

Tabela 5.10: Floresta aleatória - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
RF 05	1.0	1.0	0.917	0.915	0.917	0.915	0.956	0.956	1.0
RF 10	1.0	1.0	0.926	0.915	0.925	0.915	0.961	0.956	1.0
RF 15	1.0	1.0	0.979	0.919	0.978	0.918	0.989	0.958	1.0
RF 20	1.0	1.0	0.999	0.927	0.998	0.926	0.999	0.962	1.0
RF 25	1.0	1.0	1.000	0.932	1.000	0.931	1.000	0.964	1.0
RF 30	1.0	1.0	1.000	0.930	1.000	0.929	1.000	0.963	1.0
RF 35	1.0	1.0	1.000	0.930	1.000	0.929	1.000	0.963	1.0
RF 40	1.0	1.0	1.000	0.930	1.000	0.929	1.000	0.963	1.0

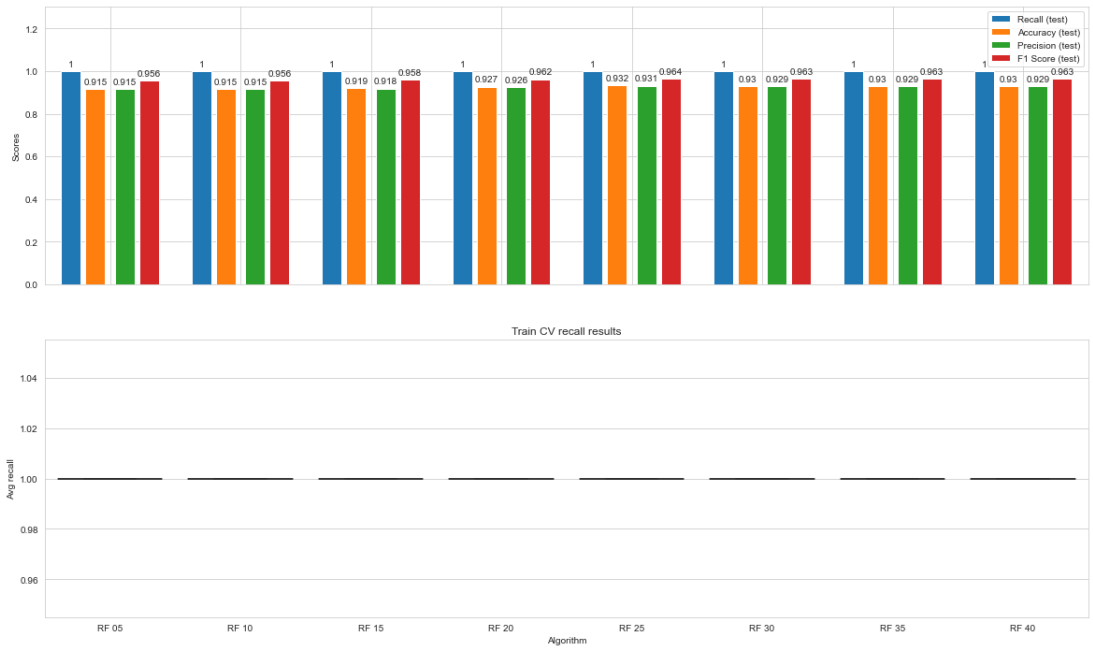

Figura 5.19: Métricas de análise - Floresta aleatória - amostra desbalanceada

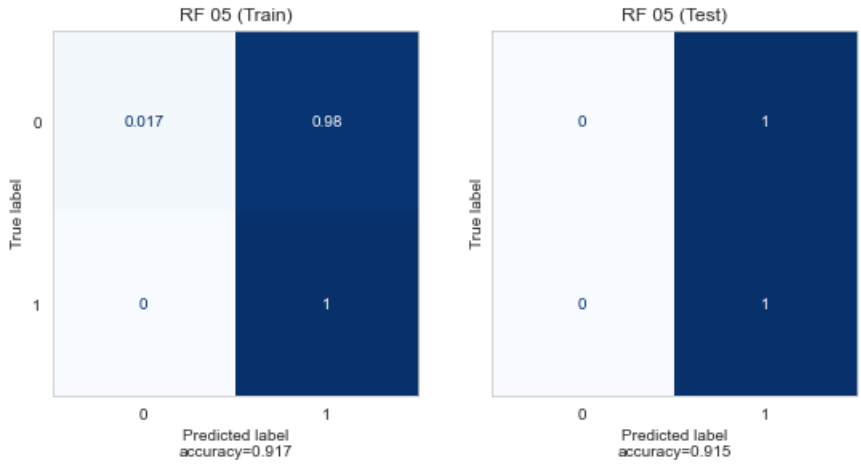

Figura 5.20: Matriz de confusão - Floresta aleatória - amostra desbalanceada

Os resultados no caso de sub-amostragem para as Florestas Aleatórias foram melhores em relação aos anteriores, apesar de *scores* baixos de revocação, 45% no grupo de teste, e na validação cruzada, 48% . O alto número de "falsos negativos"é preocupante, 55% das amostras com dano foram classificadas como sãs no grupo de teste, Figura 5.22.

Tabela 5.11: Floresta aleatória - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
RF 20	1.000	0.453	1.000	0.476	1.000	0.947	1.000	0.613	0.480
RF 25	1.000	0.451	1.000	0.476	1.000	0.950	1.000	0.612	0.489
RF 30	1.000	0.449	1.000	0.474	1.000	0.950	1.000	0.610	0.489
RF 35	1.000	0.449	1.000	0.474	1.000	0.950	1.000	0.610	0.489
RF 40	1.000	0.449	1.000	0.474	1.000	0.950	1.000	0.610	0.489
RF 15	1.000	0.444	1.000	0.469	1.000	0.950	1.000	0.605	0.497
RF 10	1.000	0.437	0.996	0.454	0.992	0.931	0.996	0.594	0.505
RF 05	0.849	0.431	0.849	0.448	0.849	0.926	0.849	0.588	0.504

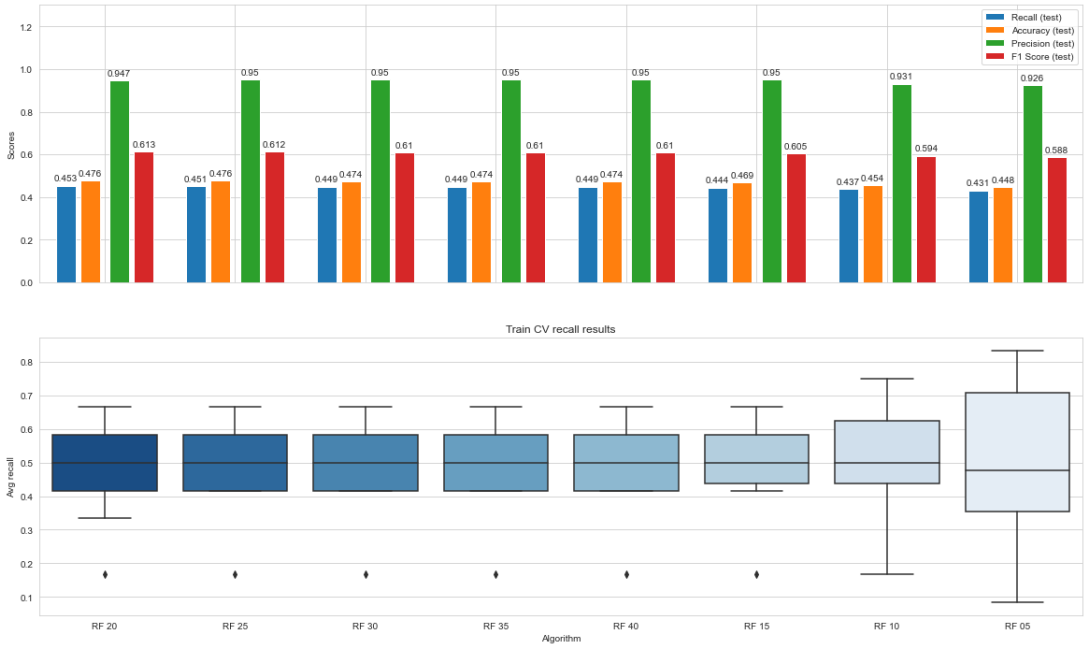

Figura 5.21: Métricas de análise - Floresta aleatória - sub-amostragem

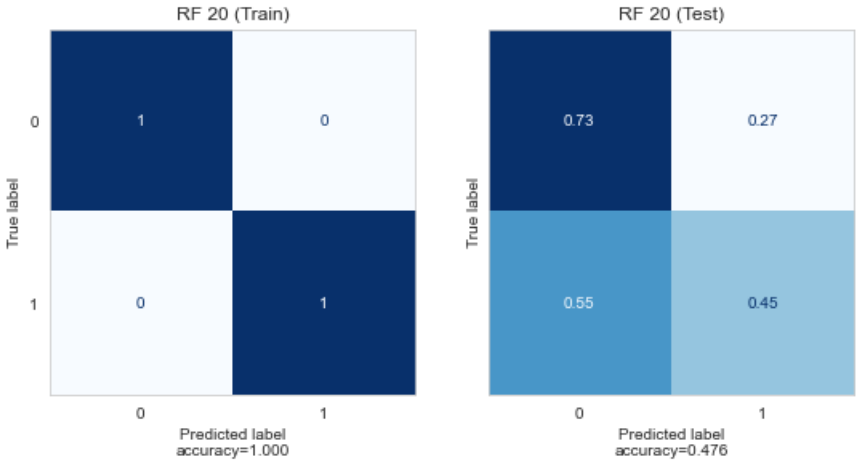

Figura 5.22: Matriz de confusão - Floresta aleatória - sub-amostragem

Assim como para as árvores de decisão, as Florestas Aleatórias foram mais performantes no caso sobre-amostrado: *scores* de revocação próximos a 94% e precisão próxima a 90% no grupo de teste, além de boa distribuição de revocações na validação cruzada nos modelos com mais de 20 de *depth*. O baixo número de "falsos negativos", 5%, é mais um ponto positivo de desempenho.

Tabela 5.12: Floresta aleatória - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
RF 30	1.000	0.944	1.000	0.899	1.000	0.946	1.000	0.945	0.925
RF 25	1.000	0.942	1.000	0.897	1.000	0.945	1.000	0.944	0.918
RF 35	1.000	0.938	1.000	0.896	1.000	0.947	1.000	0.943	0.926
RF 40	1.000	0.937	1.000	0.892	1.000	0.945	1.000	0.941	0.929
RF 20	1.000	0.924	1.000	0.877	1.000	0.941	1.000	0.932	0.908
RF 15	0.993	0.897	0.995	0.851	0.996	0.938	0.995	0.917	0.858
RF 10	0.867	0.755	0.902	0.718	0.933	0.923	0.899	0.831	0.745
RF 05	0.647	0.600	0.643	0.577	0.642	0.907	0.645	0.722	0.605

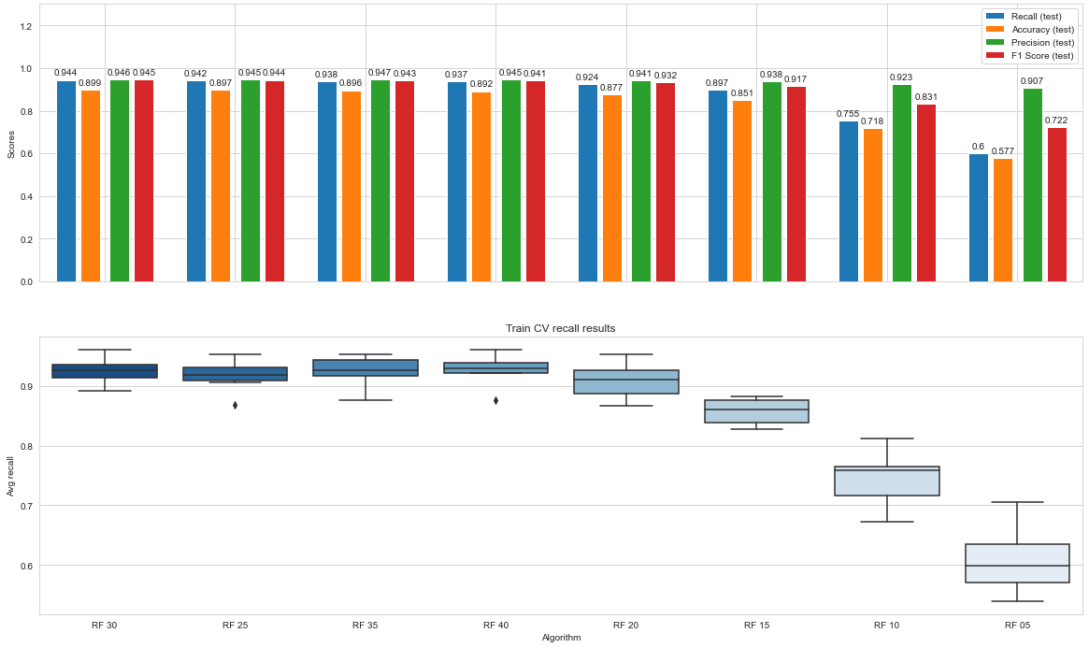

Figura 5.23: Métricas de análise - Floresta aleatória - sobre-amostragem

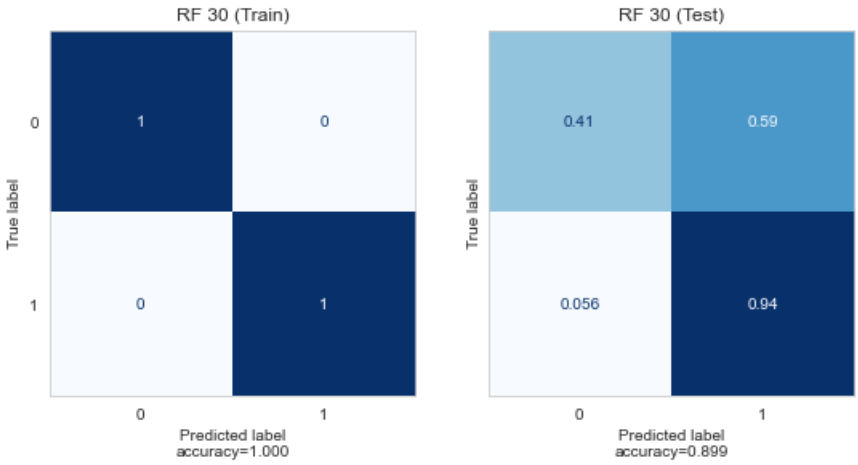

Figura 5.24: Matriz de confusão - Floresta aleatória - sobre-amostragem

Os vetores de máquinas de suporte (SVM) foram os melhores modelos para o caso de treinamento com dados desbalanceados. As performances de 66% de revocação no grupo de teste e 66% de média de revocação juntamente boa distribuição na validação cruzada nos modelos SVM 01 e 02, Figura 5.25 são os destaques.

Tabela 5.13: Support Vector Machines - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
SVM 01	0.668	0.661	0.645	0.642	0.922	0.926	0.775	0.772	0.668
SVM 02	0.668	0.661	0.645	0.642	0.922	0.926	0.775	0.772	0.668
SVM 05	0.668	0.661	0.645	0.642	0.922	0.926	0.775	0.772	0.613
SVM 06	0.668	0.661	0.645	0.642	0.922	0.926	0.775	0.772	0.613
SVM 07	0.497	0.538	0.499	0.526	0.918	0.905	0.645	0.675	0.504
SVM 08	0.497	0.538	0.499	0.526	0.918	0.905	0.645	0.675	0.504
SVM 04	0.376	0.429	0.400	0.444	0.924	0.922	0.534	0.586	0.412
SVM 03	0.341	0.380	0.371	0.403	0.924	0.921	0.498	0.538	0.410

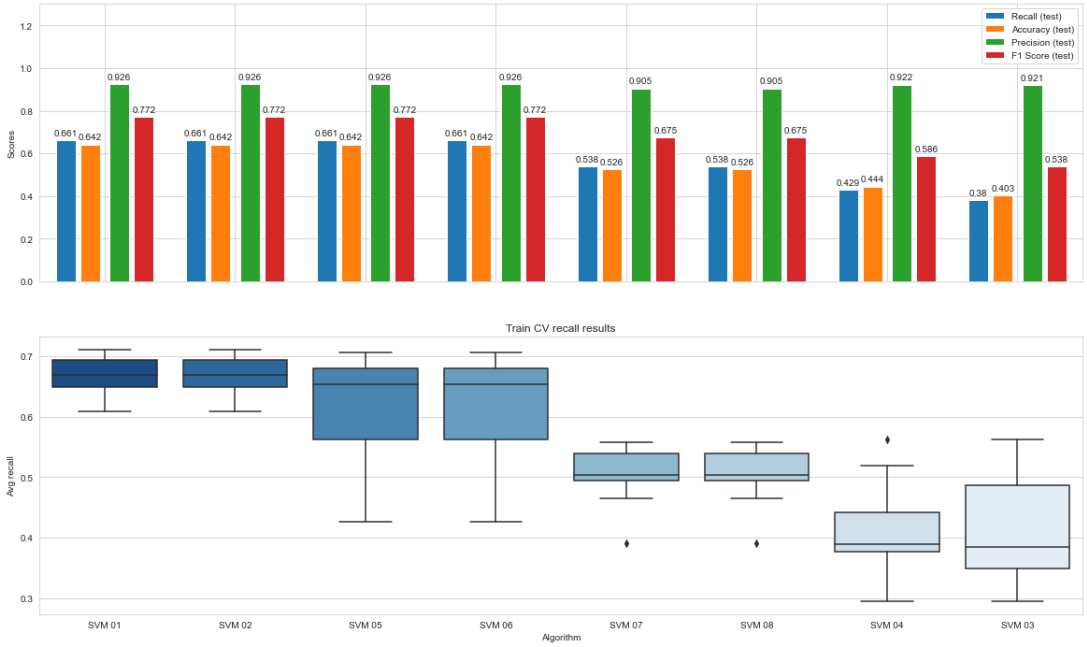

Figura 5.25: Métricas de análise - Support Vector Machines - amostra desbalanceada

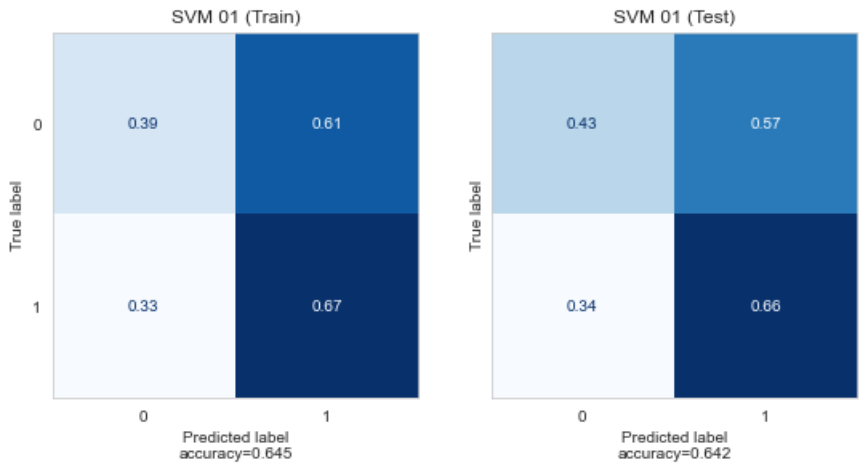

Figura 5.26: Matriz de confusão - Support Vector Machines - amostra desbalanceada

No caso de sub-amostragem os modelos SVM tiveram um desempenho melhor em relação ao caso de amostras desbalanceadas cerca de 75% de revocação no grupo de teste, 81% de média de revocação e boa distribuição na validação cruzada para os modelos SVM 03 e 04, Figura 5.27.

Tabela 5.14: Support Vector Machines - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
SVM 03	0.824	0.757	0.542	0.721	0.527	0.925	0.643	0.833	0.814
SVM 04	0.824	0.757	0.542	0.721	0.527	0.925	0.643	0.833	0.814
SVM 01	0.689	0.661	0.538	0.642	0.529	0.926	0.599	0.772	0.690
SVM 02	0.689	0.661	0.538	0.642	0.529	0.926	0.599	0.772	0.690
SVM 07	0.387	0.496	0.475	0.493	0.469	0.907	0.424	0.642	0.392
SVM 08	0.387	0.496	0.475	0.493	0.469	0.907	0.424	0.642	0.392
SVM 05	0.538	0.438	0.521	0.454	0.520	0.927	0.529	0.595	0.581
SVM 06	0.538	0.438	0.521	0.454	0.520	0.927	0.529	0.595	0.581

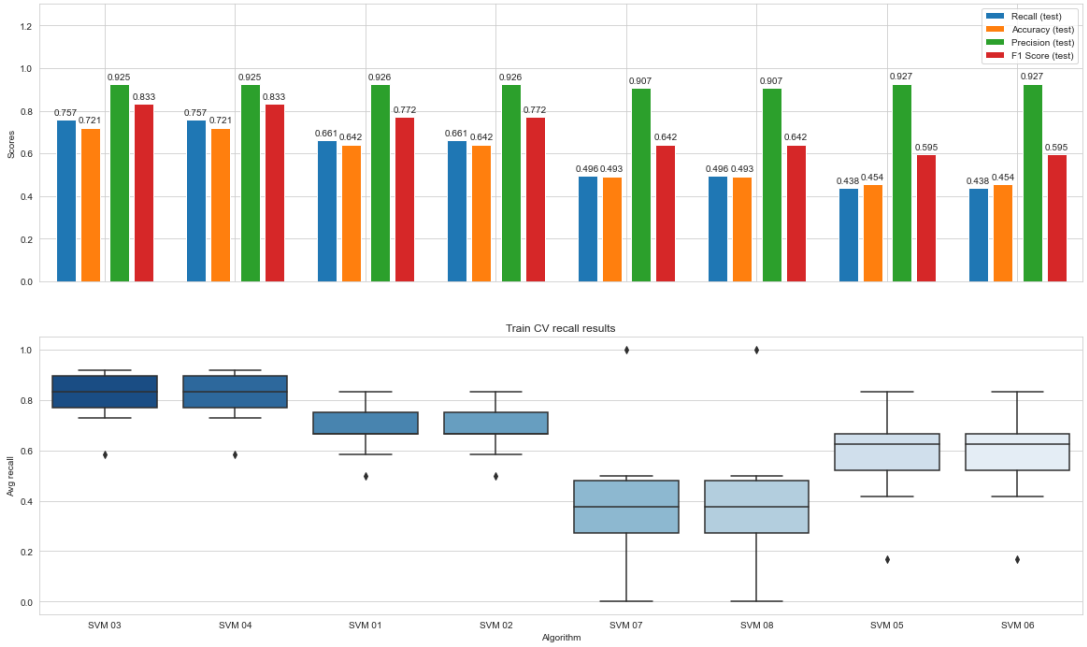

Figura 5.27: Métricas de análise - Support Vector Machines - sub-amostragem

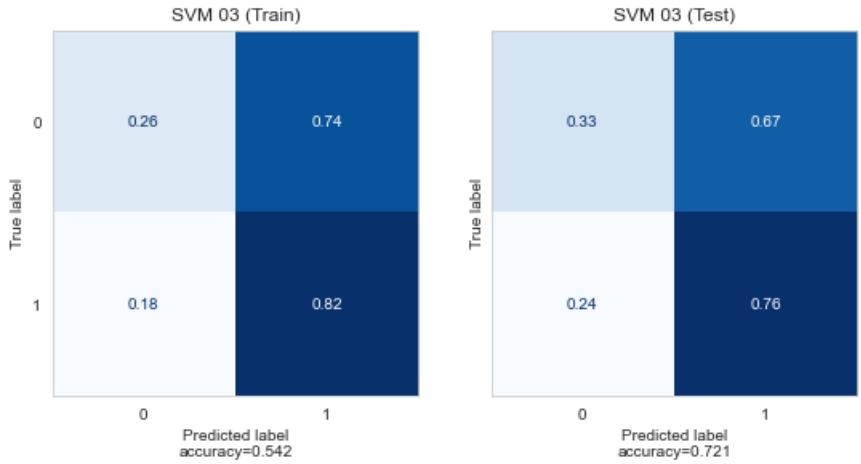

Figura 5.28: Matriz de confusão - Support Vector Machines - sub-amostragem

O caso de sobre-amostragem teve um resultado semelhante ao caso desbalanceado, com distribuições melhores na validação cruzada para os modelos SVM 03, 04, 05 e 06 do que na amostragem desbalanceada.

Tabela 5.15: Support Vector Machines - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
SVM 01	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 02	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 03	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 04	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 05	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 06	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM 07	0.497	0.538	0.493	0.526	0.493	0.905	0.495	0.675	0.499
SVM 08	0.497	0.538	0.493	0.526	0.493	0.905	0.495	0.675	0.499

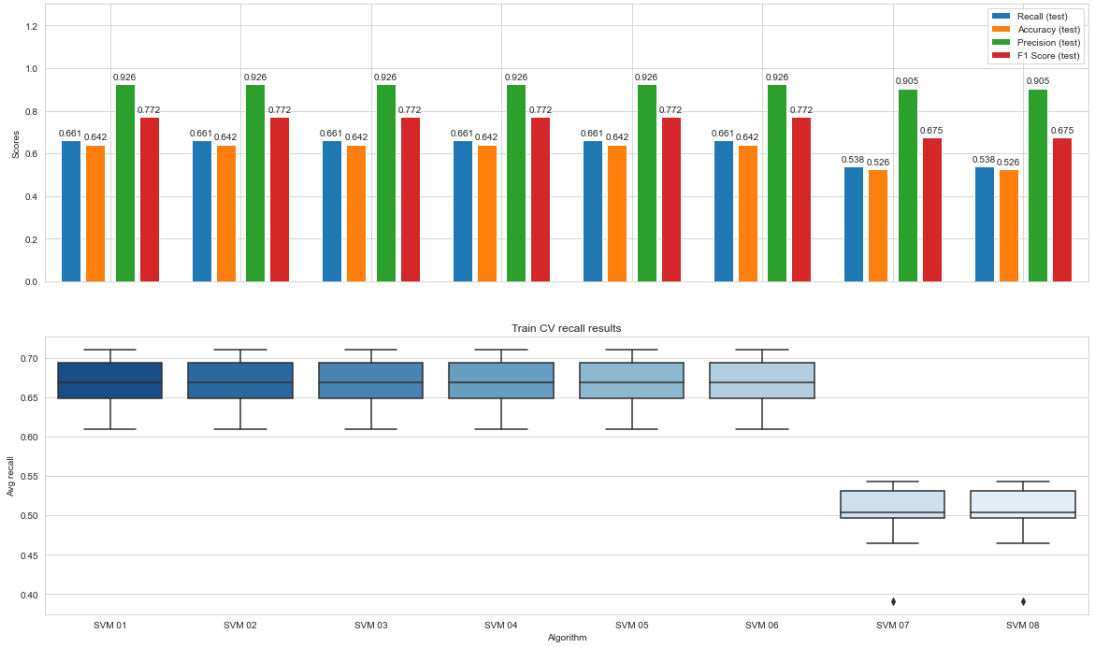

Figura 5.29: Métricas de análise - Support Vector Machines - sobre-amostragem

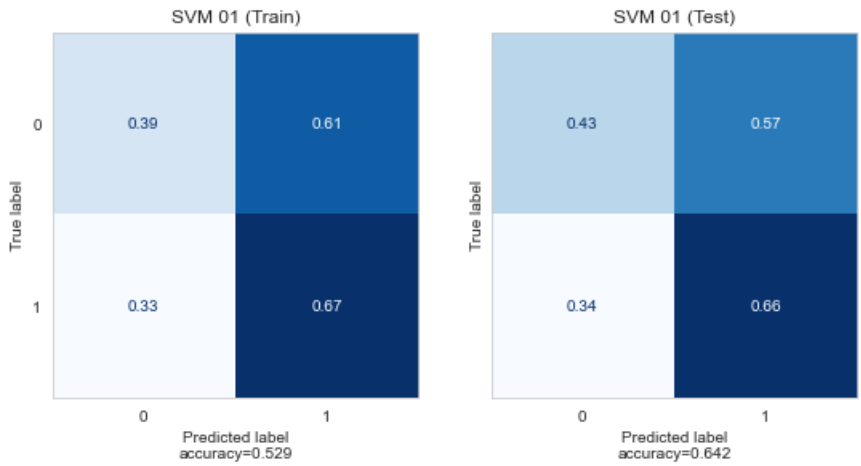

Figura 5.30: Matriz de confusão - Support Vector Machines - sobre-amostragem

O caso de amostragem desbalanceada não teve desempenho razoável para as redes neurais artificiais, e classificaram todos os exemplos de viga como "danificadas".

Tabela 5.16: Redes neurais artificiais - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
MLP 01	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 02	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 03	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 04	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 05	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 06	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 07	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0
MLP 08	1.0	1.0	0.915	0.915	0.915	0.915	0.956	0.956	1.0

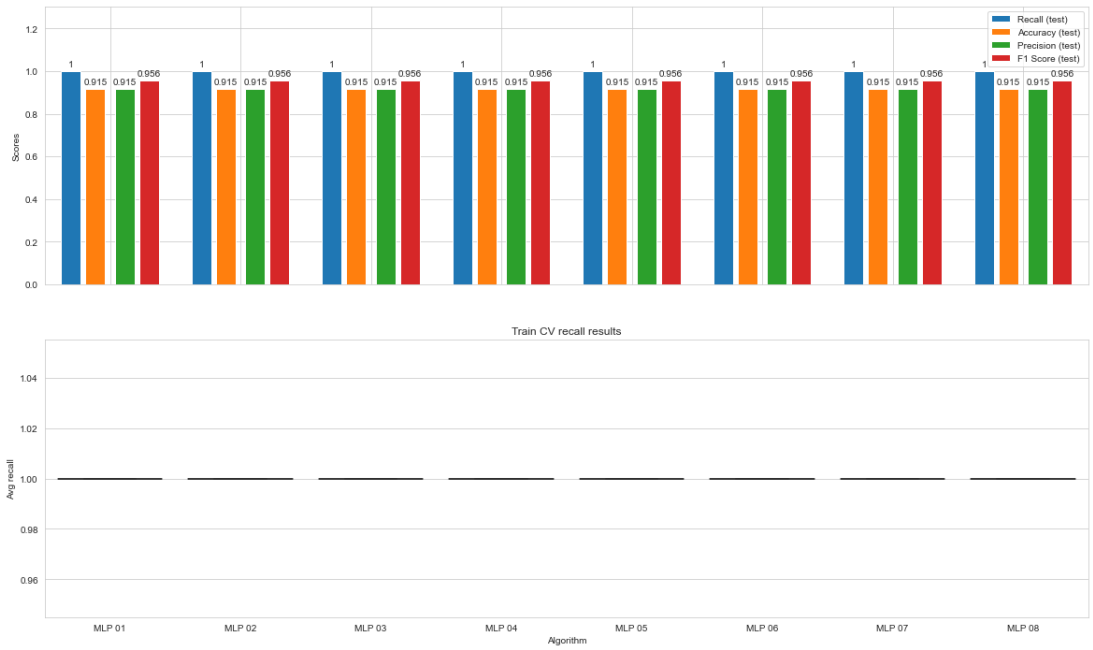

Figura 5.31: Métricas de análise - Redes neurais artificiais - amostra desbalanceada

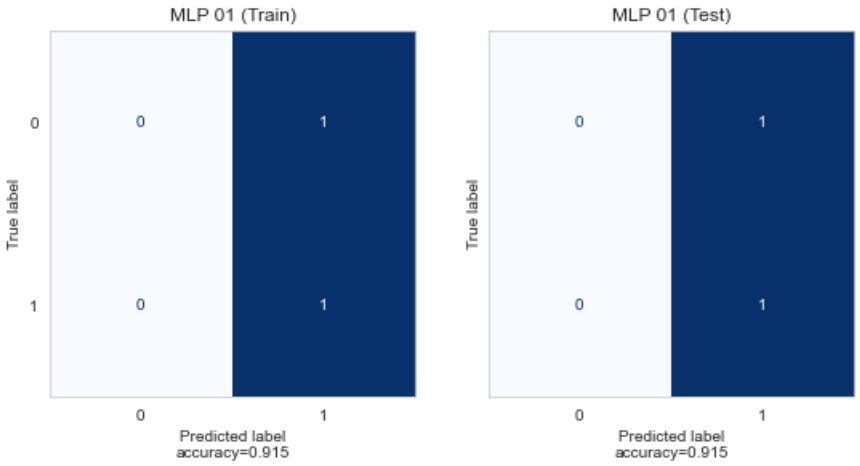

Figura 5.32: Matriz de confusão - Redes neurais artificiais - amostra desbalanceada

A sub-amostragem teve um ganho de confiabilidade em relação a amostragem desbalanceada, porém apesar do *score* alto de revocação, 90%, no grupo de teste a distribuição da validação cruzada foi esparsa para o modelo MLP 04.

Tabela 5.17: Redes neurais artificiais - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
MLP 04	0.866	0.904	0.496	0.836	0.498	0.916	0.632	0.910	0.483
MLP 03	0.496	0.630	0.475	0.604	0.476	0.909	0.486	0.744	0.444
MLP 01	0.647	0.509	0.555	0.514	0.546	0.927	0.592	0.657	0.598
MLP 05	0.513	0.413	0.550	0.434	0.555	0.931	0.533	0.572	0.439
MLP 07	0.311	0.339	0.462	0.358	0.446	0.895	0.366	0.491	0.310
MLP 06	0.437	0.328	0.529	0.360	0.536	0.923	0.481	0.484	0.261
MLP 02	0.353	0.308	0.492	0.343	0.488	0.924	0.410	0.462	0.251
MLP 08	0.143	0.158	0.508	0.214	0.531	0.906	0.225	0.269	0.298

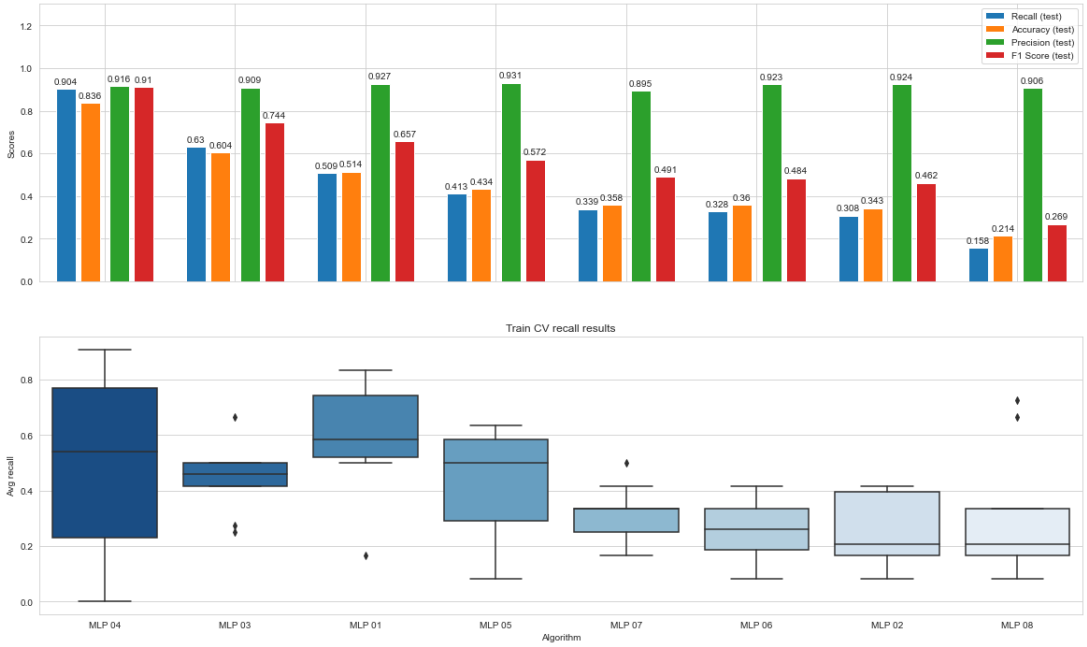

Figura 5.33: Métricas de análise - Redes neurais artificiais - sub-amostragem

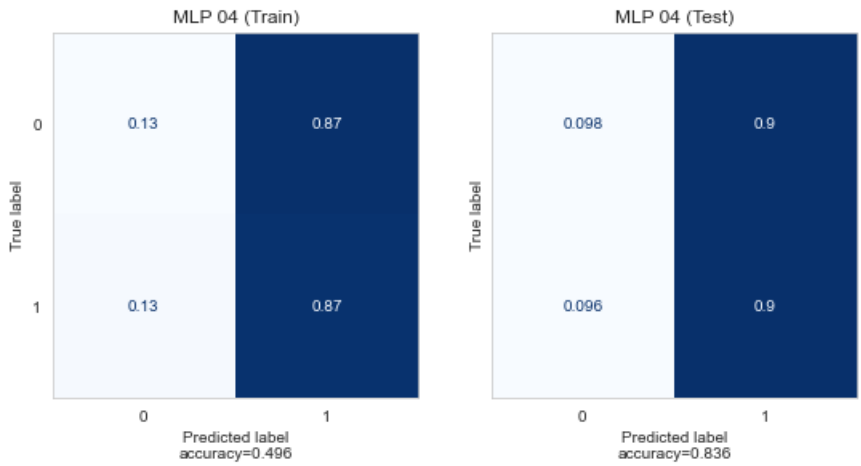

Figura 5.34: Matriz de confusão - Redes neurais artificiais - sub-amostragem

Os resultados com dados de sobre-amostragem foram os melhores para as redes neurais artificiais, chegando a 66% de revocação no grupo de teste e de média de validação cruzada, que ficou bem distribuída no modelo MLP 02, Figura 5.35.

Tabela 5.18: Redes neurais artificiais - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
MLP 02	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
MLP 05	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
MLP 06	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
MLP 04	0.584	0.598	0.532	0.582	0.529	0.917	0.555	0.724	0.608
MLP 08	0.588	0.587	0.533	0.577	0.530	0.923	0.558	0.718	0.619
MLP 03	0.479	0.538	0.509	0.532	0.509	0.917	0.493	0.678	0.466
MLP 01	0.558	0.529	0.518	0.529	0.517	0.924	0.536	0.673	0.565
MLP 07	0.407	0.466	0.518	0.476	0.522	0.924	0.458	0.619	0.544

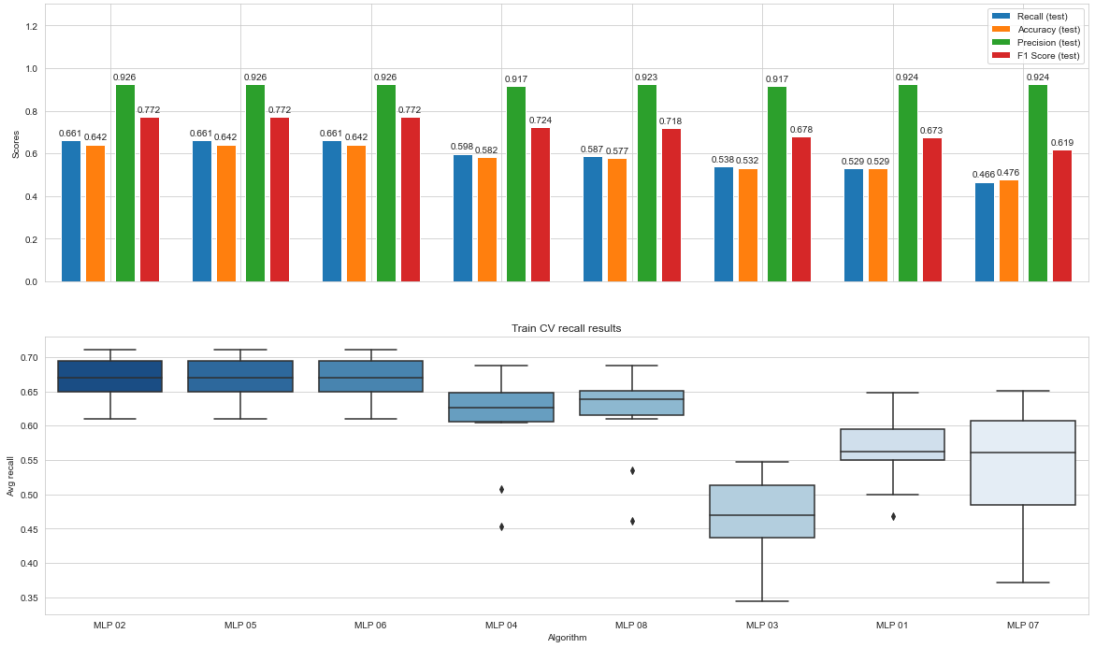

Figura 5.35: Métricas de análise - Redes neurais artificiais - sobre-amostragem

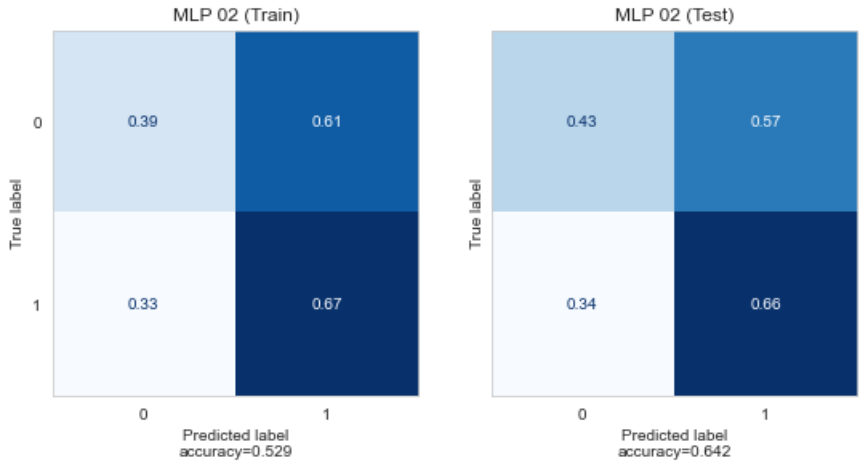

Figura 5.36: Matriz de confusão - Redes neurais artificiais - sobre-amostragem

Os modelos KNN classificaram praticamente 100% das amostras como "danificadas", inviabilizando uma análise qualitativa para o caso de amostragem desbalanceada.

Tabela 5.19: K-nearest neighbor - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
KNN 02	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	0.998
KNN 03	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	0.999
KNN 04	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	1.000
KNN 05	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	1.000
KNN 06	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	1.000
KNN 07	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	1.000
KNN 08	1.0	1.000	1.0	0.915	1.0	0.915	1.0	0.956	1.000
KNN 01	1.0	0.991	1.0	0.912	1.0	0.919	1.0	0.954	0.984

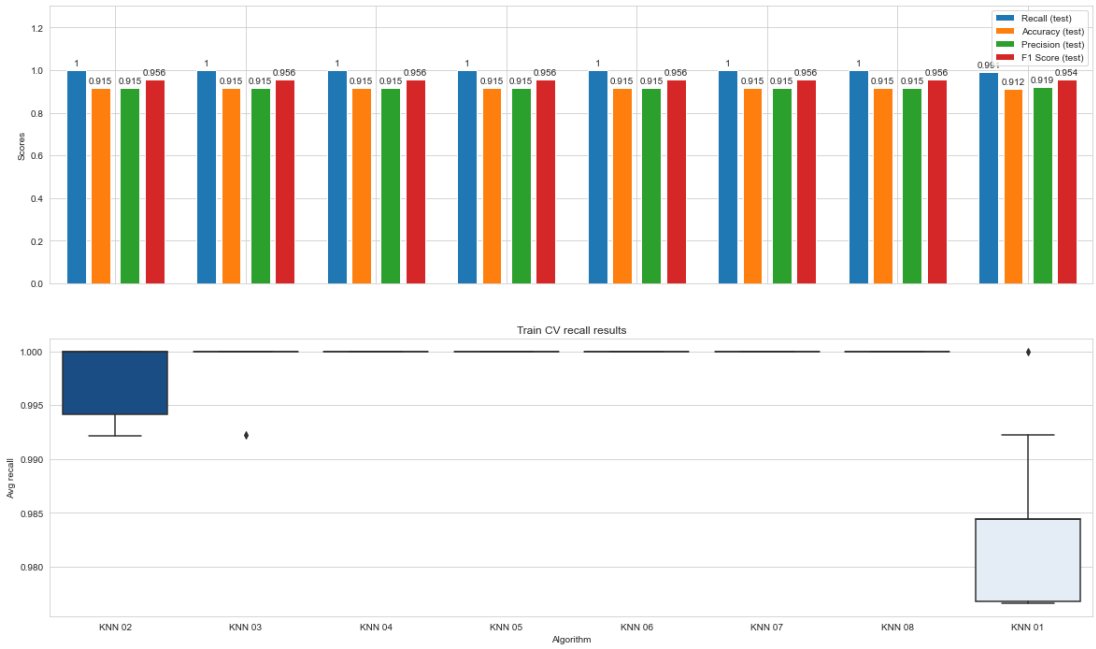

Figura 5.37: Métricas de análise - K-nearest neighbor - amostra desbalanceada

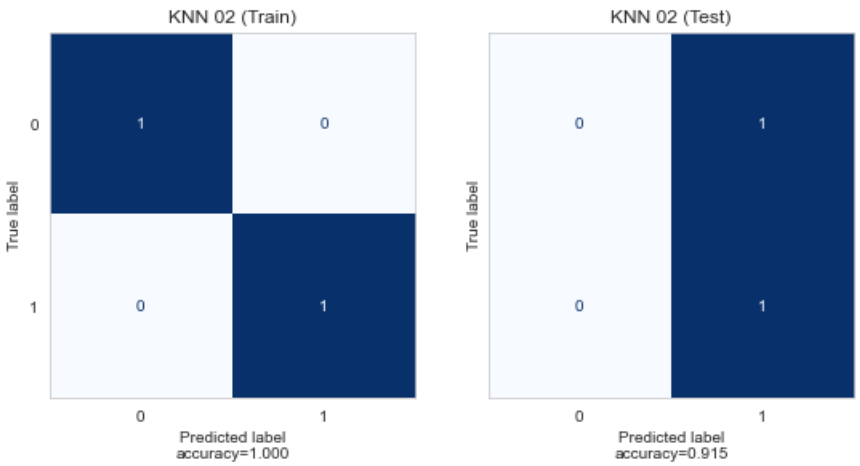

Figura 5.38: Matriz de confusão - K-nearest neighbor - amostra desbalanceada

Apesar de permitir uma análise melhor, a sub-amostragem levou a um desempenho de 50% de revocação no grupo de teste e 53% de média de revocação na validação cruzada. Destaca-se negativamente a alta quantidade de "falsos negativos"no grupo de teste: 49% para o modelo KNN 01.

Tabela 5.20: K-nearest neighbor - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
KNN 01	1.0	0.507	1.0	0.504	1.0	0.912	1.0	0.652	0.530
KNN 03	1.0	0.500	1.0	0.493	1.0	0.902	1.0	0.643	0.539
KNN 02	1.0	0.484	1.0	0.481	1.0	0.905	1.0	0.630	0.564
KNN 04	1.0	0.482	1.0	0.478	1.0	0.902	1.0	0.628	0.556
KNN 05	1.0	0.482	1.0	0.478	1.0	0.902	1.0	0.628	0.556
KNN 06	1.0	0.482	1.0	0.478	1.0	0.902	1.0	0.628	0.556
KNN 07	1.0	0.482	1.0	0.478	1.0	0.902	1.0	0.628	0.556
KNN 08	1.0	0.482	1.0	0.478	1.0	0.902	1.0	0.628	0.556

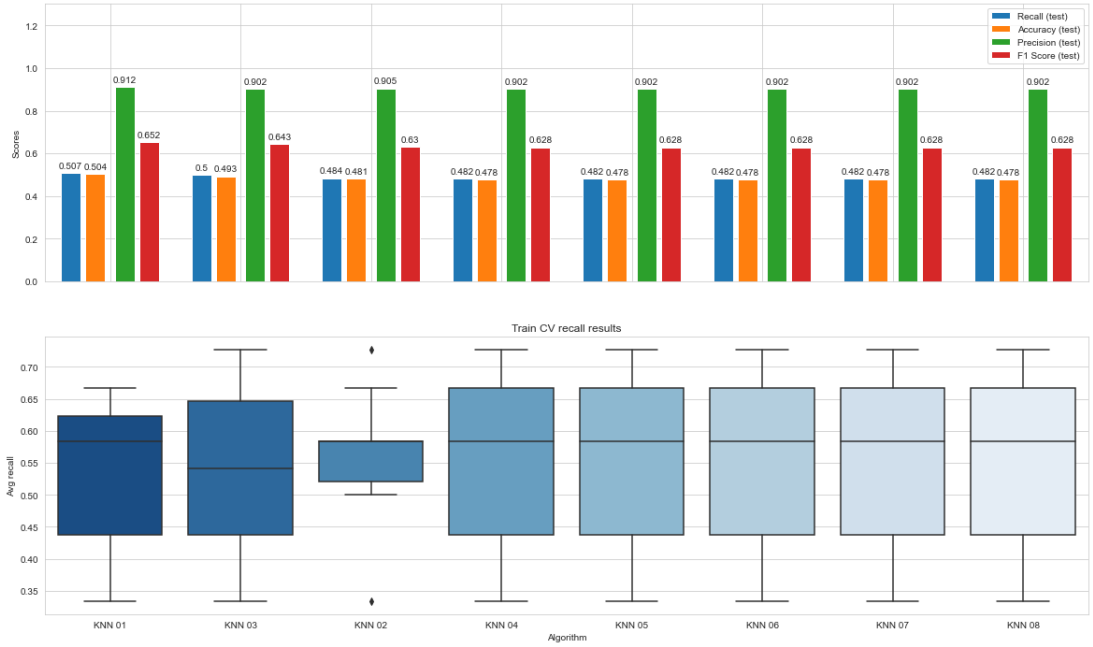

Figura 5.39: Métricas de análise - K-nearest neighbor - sub-amostragem

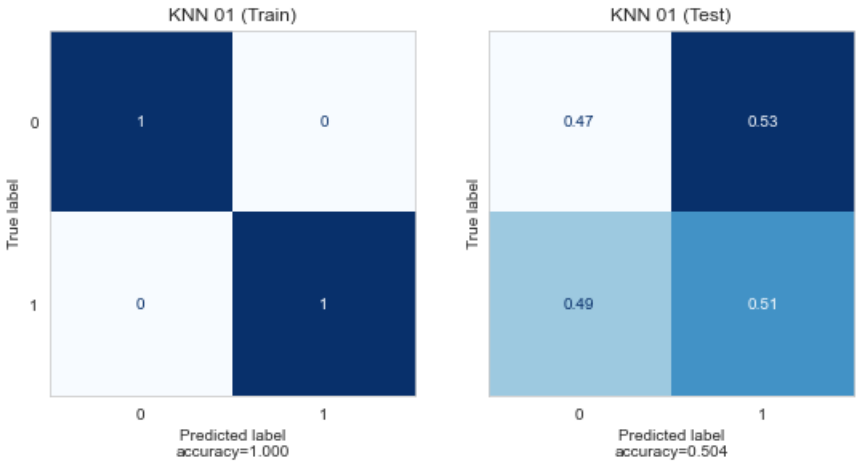

Figura 5.40: Matriz de confusão - K-nearest neighbor - sub-amostragem

O caso sobre-amostrado mostrou-se o mais performante dentro dos modelos KNN. O modelo KNN 04 atingiu 93% de revocação e 89% de média de revocação na validação cruzada, com boa distribuição, e uma quantidade pequena de "falsos negativos", Figuras 5.41 e 5.42.

Tabela 5.21: K-nearest neighbor - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
KNN 04	1.0	0.935	1.0	0.859	1.0	0.913	1.0	0.924	0.893
KNN 05	1.0	0.933	1.0	0.856	1.0	0.912	1.0	0.922	0.861
KNN 07	1.0	0.918	1.0	0.842	1.0	0.910	1.0	0.914	0.865
KNN 08	1.0	0.918	1.0	0.842	1.0	0.910	1.0	0.914	0.866
KNN 03	1.0	0.909	1.0	0.837	1.0	0.913	1.0	0.911	0.883
KNN 02	1.0	0.897	1.0	0.826	1.0	0.912	1.0	0.904	0.871
KNN 06	1.0	0.893	1.0	0.821	1.0	0.910	1.0	0.901	0.844
KNN 01	1.0	0.886	1.0	0.818	1.0	0.912	1.0	0.899	0.851

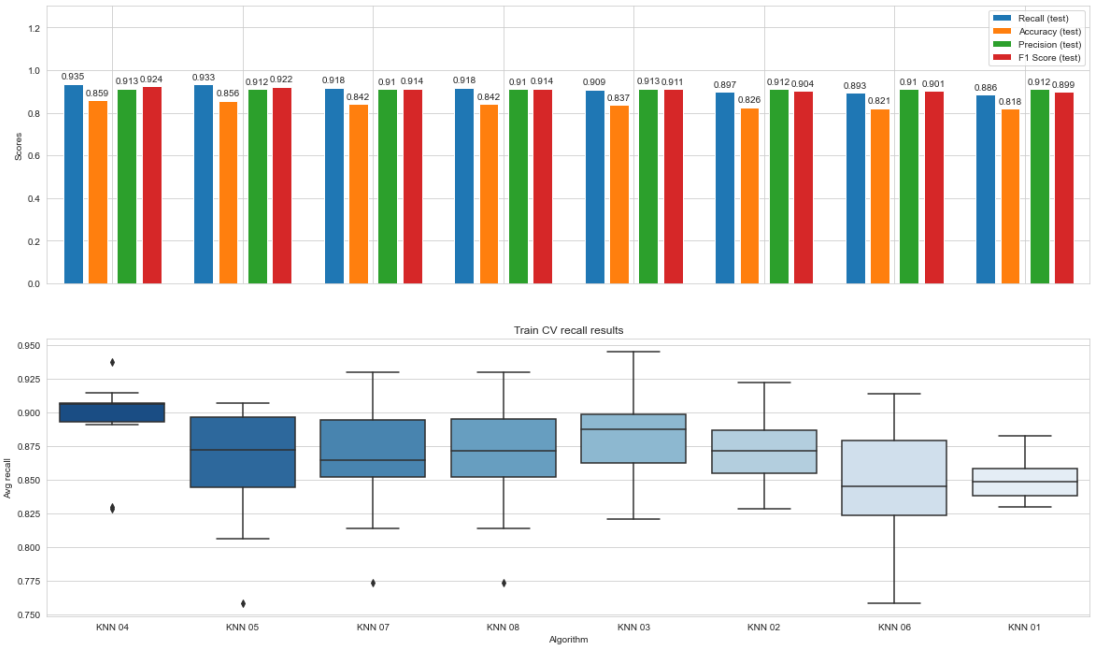

Figura 5.41: Métricas de análise - K-nearest neighbor - sobre-amostragem

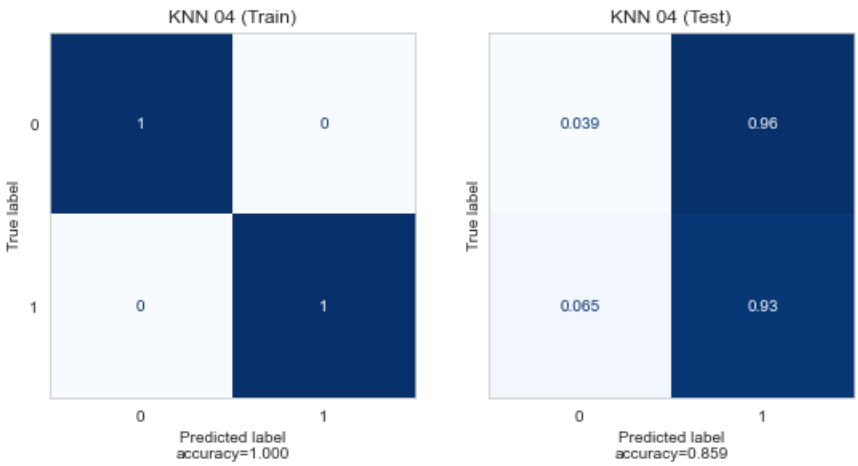

Figura 5.42: Matriz de confusão - K-nearest neighbor - sobre-amostragem

A comparação de modelos bem performantes dentro quando treinados em amostras desbalnceadas é contra-intuitiva, apesar de valores menores de *scores* o modelo SVM tem o resultado mais confiável pois não sofreu do viés do desbalanceamento de amostras, conforme pode ser observado nas Figuras 5.43 e 5.44.

Tabela 5.22: Comparação de modelos - amostra desbalanceada - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
LINr	1.000	1.000	0.915	0.915	0.915	0.915	0.956	0.956	1.000
LOGr	1.000	1.000	0.915	0.915	0.915	0.915	0.956	0.956	1.000
RF	1.000	1.000	1.000	0.930	1.000	0.929	1.000	0.963	1.000
MLP	1.000	1.000	0.915	0.915	0.915	0.915	0.956	0.956	1.000
KNN	1.000	1.000	1.000	0.915	1.000	0.915	1.000	0.956	0.999
DT	1.000	0.996	0.924	0.914	0.923	0.917	0.960	0.955	0.998
SVM	0.668	0.661	0.645	0.642	0.922	0.926	0.775	0.772	0.668

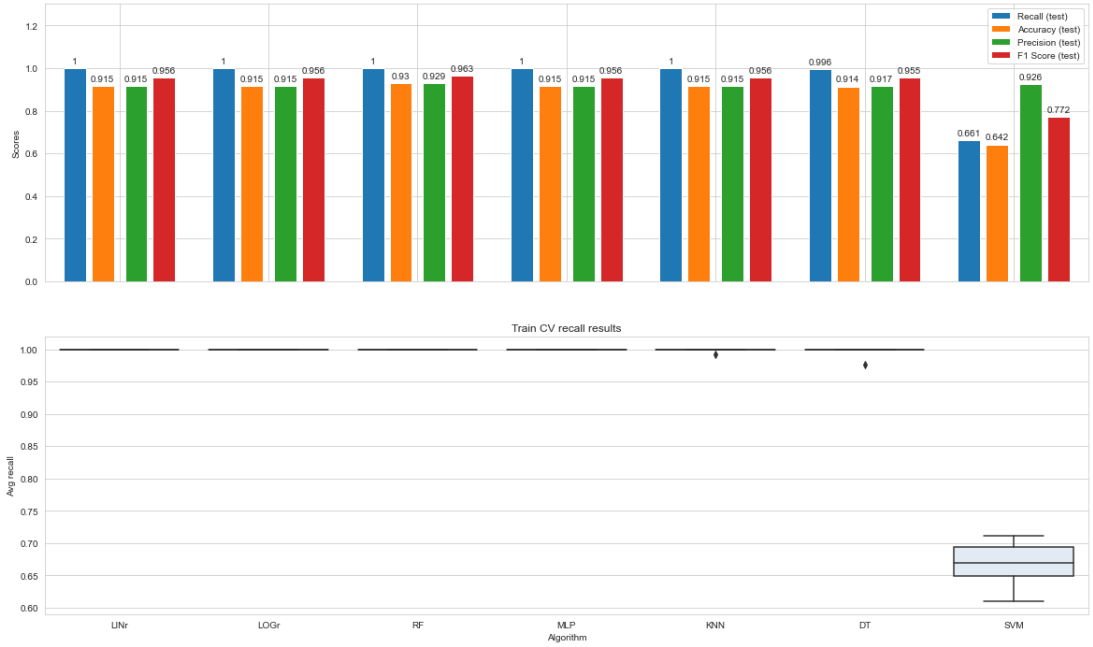

Figura 5.43: Métricas de análise - Comparação de modelos - amostra desbalanceada

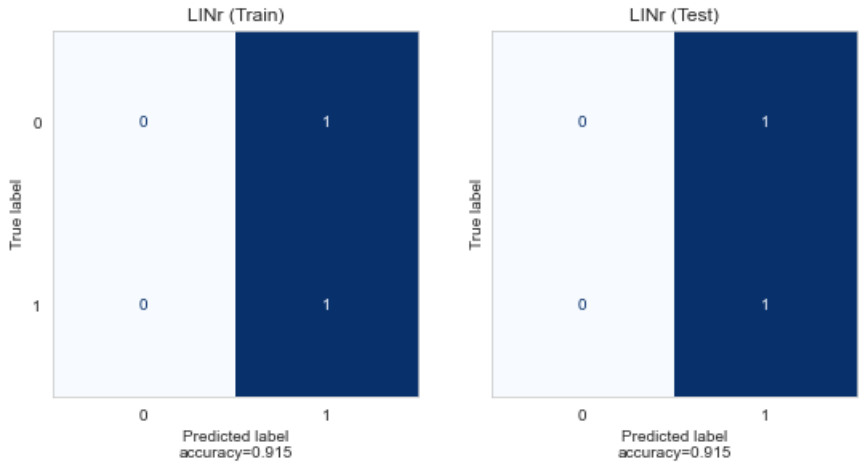

Figura 5.44: Matriz de confusão - Comparação de modelos - amostra desbalanceada

Novamente, a comparação de performance entre modelos no caso de subamostragem requer uma análise mais profunda pois, apesar de ter bons *scores* o modelo MLP quando analisado por validação cruzada se mostrou inconfiável enquanto Árvores de Decisão e SVM se mostraram mais viáveis, conforme pode ser observado na Figura 5.45.

Tabela 5.23: Comparação de modelos - sub-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
MLP	0.866	0.904	0.496	0.836	0.498	0.916	0.632	0.910	0.483
DT	0.950	0.808	0.630	0.756	0.579	0.916	0.720	0.859	0.640
SVM	0.824	0.757	0.542	0.721	0.527	0.925	0.643	0.833	0.814
LOGr	0.597	0.569	0.521	0.566	0.518	0.929	0.555	0.706	0.622
LINr	0.571	0.540	0.525	0.542	0.523	0.931	0.546	0.683	0.564
KNN	1.000	0.507	1.000	0.504	1.000	0.912	1.000	0.652	0.530
RF	1.000	0.437	0.996	0.454	0.992	0.931	0.996	0.594	0.505

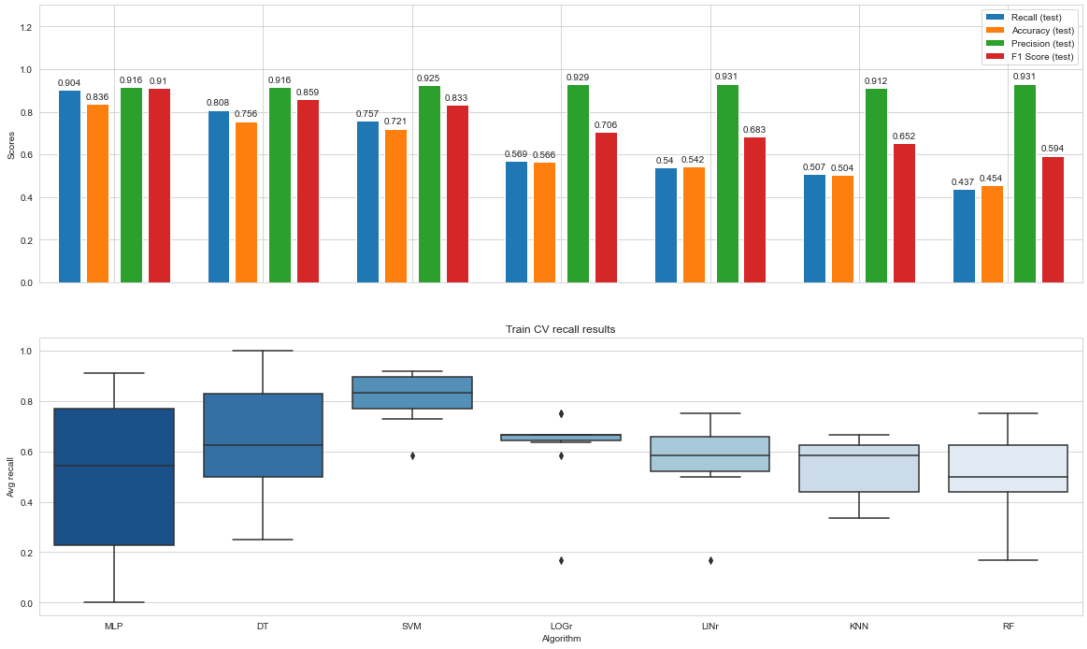

Figura 5.45: Métricas de análise - Comparação de modelos - sub-amostragem

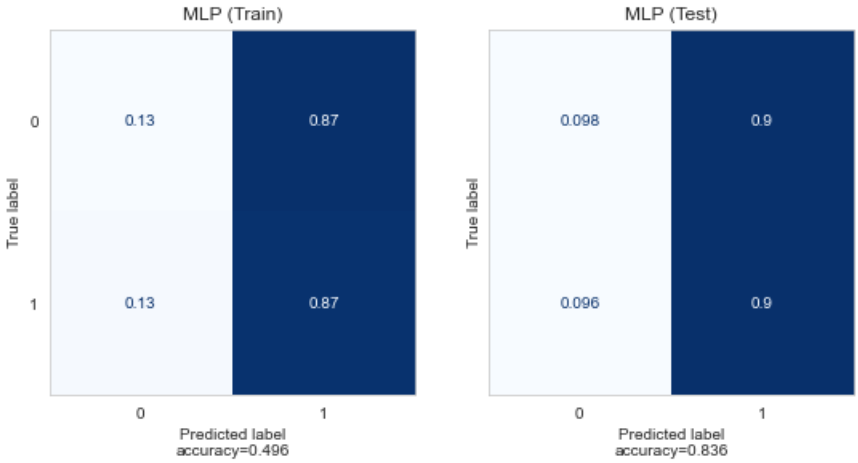

Figura 5.46: Matriz de confusão - Comparação de modelos - sub-amostragem

Por fim, no caso de sobre-amostragem três modelos se destacaram dos demais em termos de desempenho: Florestas Aleatórias, KNN e Árvores de Decisão,Figura 5.47.

Tabela 5.24: Comparação de modelos - sobre-amostragem - resultados

Name	Recall (train)	Recall (test)	Acc (train)	Acc (test)	Precision (train)	Precision (test)	F1 Score (train)	F1 Score (test)	CV Recall (train)
RF	1.000	0.944	1.000	0.899	1.000	0.946	1.000	0.945	0.925
KNN	1.000	0.935	1.000	0.859	1.000	0.913	1.000	0.924	0.893
DT	1.000	0.889	1.000	0.837	1.000	0.930	1.000	0.909	0.873
LINr	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
LOGr	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
SVM	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668
MLP	0.668	0.661	0.529	0.642	0.523	0.926	0.587	0.772	0.668

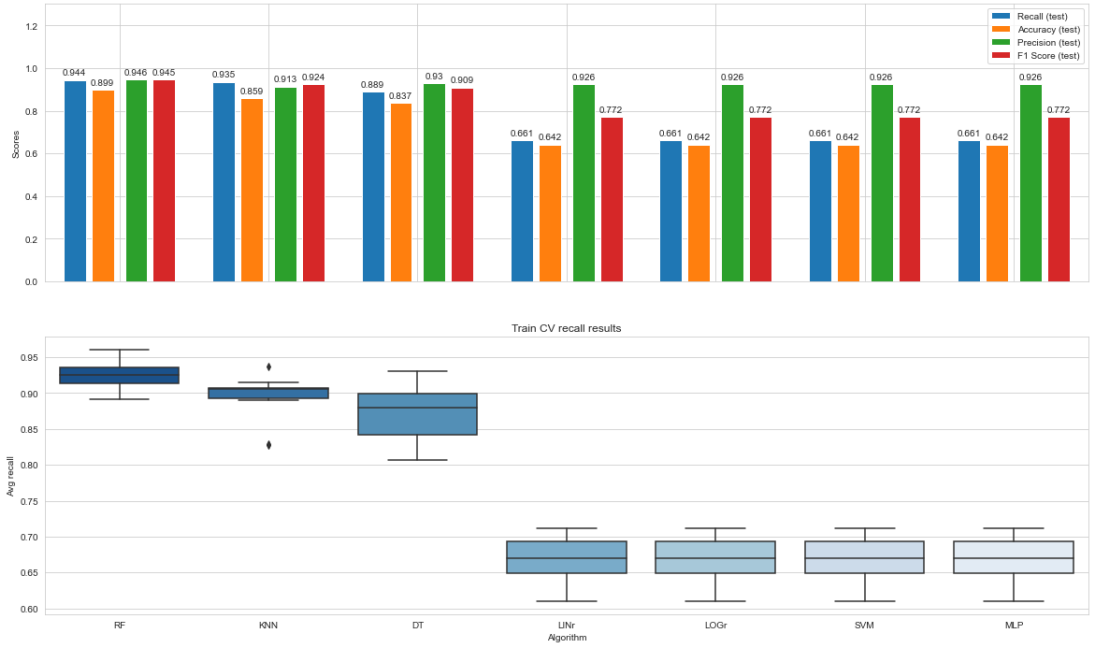

Figura 5.47: Métricas de análise - Comparação de modelos - sobre-amostragem

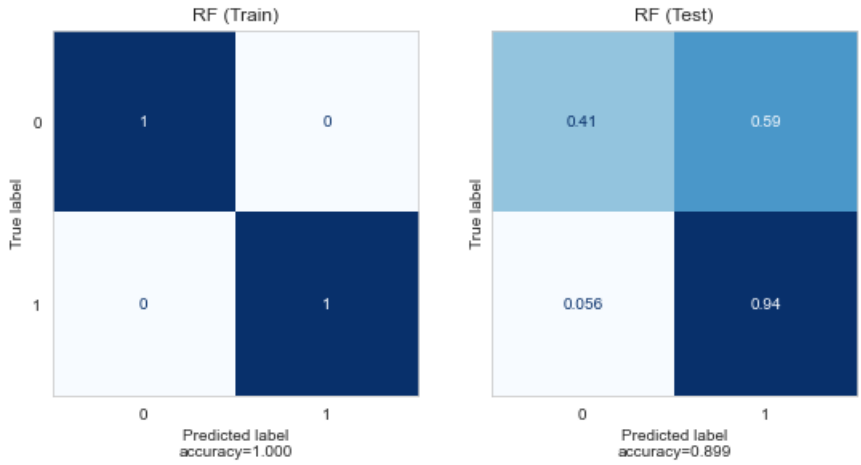

Figura 5.48: Matriz de confusão - Comparação de modelos - sobre-amostragem

6
Conclusão

O pre-processamento do banco de dados se mostrou indispensável no problema de identificação de danos com dados de vigas numéricas. Especificamente neste trabalho os métodos de aprendizado de máquinas para classificação baseados em Regressão Linear, Regressão Logística, Árvore de Decisão, Florestas Aleatórias, Redes Neurais Artificiais e *K-nearest neighbors* quando treinados com uma base de dados desbalanceada performaram insatisfatoriamente, em contra partida *Support Vector Machines* se mostrou viável neste cenário.

Os algoritmos de aprendizado de máquinas tiveram um desempenho melhor no caso de sub-amostragem do que o treinamento desbalanceado, com exceção do método SVM, que teve uma queda de performance. A sobre-amostragem permitiu o melhor desempenho dentro das três hipóteses de bancos de dados apresentadas, neste cenário três métodos se destacaram positivamente: Florestas Aleatórias, KNN e Árvores de Decisão.

Apesar da robustez e sofisticação que os métodos de *Machine Learning* proporcionam nos problemas de engenharia, os benefícios são limitados pela qualidade do banco de dados em que são treinados ou aplicados. Apesar de *scores* elevados apresentados neste trabalhos, o desempenho dos mesmos métodos quando alimentados com dados experimentais de vibrações deve ser diferente, uma vez que estão sujeitos a ruídos de diferentes fontes. Estas dificuldades certamente também devem ser estudadas e, possivelmente, superadas com uma combinação de métodos de ML para preparação dos dados ou mesmo a utilização de outros modelos mais robustos.

Por fim, o Aprendizado de Máquinas é uma ferramenta que pode ser utilizada no problema de identificação de danos em estruturas. Além disso, sua utilização pode ser explorada como uma forma de enriquecer bancos de dados experimentais, que costumam ser escassos. Portanto, é importante aprofundar o estudo dessa área buscando soluções que possam contribuir para a melhoria da identificação de danos em estruturas.

7
Referências bibliográficas

ALLEMANG, R.; BROWN, D. A correlation coefficient for modal vector analysis. **IMAC**, v. 1, n. 1, p. 110–116, 1982.

AVCI OSAMA ABDELJABER, S. K. M. H. M. G. D. J. I. O. A review of vibration-based damage detection in civil structures: From traditional methods to machine learning and deep learning applications. **Mechanical Systems and Signal Processing**, v. 147, 2021.

CHANG, C.-W. K. K. Modal-parameter identification and vibration-based damage detection of a damaged steel truss bridge. **Engineering Structures**, v. 122, p. 156–173, 2016.

CHAVES, A. da C. F. **Extração de regras fuzzy para máquinas de vetor suporte (SVM) para classificação em múltiplas classes**. 225 p. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

CHAWLA, N. V. et al. Smote: Synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, v. 1, n. 16, p. 321–357, 2002.

CLéMENT, A. **Détection de nouveauté pour le monitoring vibratoire des structures de génie civil: Approches chaotique et statique de l'extration d'indicateurs**. 200 p. Tese (Doutorado) — L'Université de Toulouse, Toulouse, 2011.

FILHO, L. A. C. M. de A. **Identificação de Estruturas em Operação através de análise modal híbrida**. 297 p. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

FRISWELL, M. I.; PENNY, J. title=,. **Structural Health Monitoring**, v. 1, p. 139–148, 2002.

GONEN, E. E. S. A hybrid method for vibration-based bridhe damage detection. **Remote Sens** — Bridge Damage Detection, 2022. Disponível em: <https://doi.org/10.3390/rs14236054>.

HE, J.; FU, Z.-F. **Modal Analysis**. 1. ed. Woburn, MA: Butterworth-Heinemann, 2001. 285 p.

KUSHWAH SUDARSAN SAHOO, A. J. K. Health monitoring of wind turbine blades through vibration signal using machine learning techniques. p. 239–247, 2021.

LE, T.-P. **Auscultation dynamique des structures à l'aide de l'analyse continue en ondelettes**. 200 p. Tese (Doutorado) — L'Université de Toulouse, Toulouse, 2011.

LE, T.-P.; ARGOUL, P. Continuous wavelet transform for modal identification using free decay response. **Journal of Sound and Vibration**, v. 277, 2003. 20 aug. de 2003. Disponível em: <http://www.elsevier.com/locate/jsvi>.

LOURENÇO, L. F. R. P. B. Dynamic identification and monitoring of cultural heritage buildings. In: . [S.l.: s.n.], 2011.

LOUZADA, D. R. **Detecção e caracterização de danos estruturais através de sensores a rede de Bragg e Redes Neurais Artificiais**. 173 p. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2013.

MEIROVITCH, L. **Fundamental of Vibrations**. [S.l.]: McGraw-Hill, 2001. 806 p. ISBN 0-07-041345-2.

NAJM, D. **Quels capteurs de vibrations pour la surveillance de santé des structures mécaniques?** 290 p. Tese (Doutorado) — L'Université Paris-Est, Paris, 2015.

NGUYEN, T. M. **Dynamique non linéaire des systèmes mécaniques couplés : Réduction de modèle et Identification**. 280 p. Tese (Doutorado) — École Nationale des Ponts et Chaussées, Paris, 2007.

PASCAL, J.-C. **Vibrations et Acoustique 2**. 1. ed. Université du Maine: École nationale superieure d'ingenieurs du Mans, 2008. 214 p.

PASCAL, J.-C. **Vibrations et Acoustique 2**. [S.l.]: ECOLE NATIONALE SUPERIEURE D'INGENIEURS DU MANS - UNIVERSITE DU MAINE, 2008.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

RYTTER, A. **Vibration based inspection of civil engineering structures**. 200 p. Tese (Doutorado) — L'Université d'Aalborg, Denmark, 1993.

SAJEDI, X. L. S. Trident: A deep learning framework for high-resolution bridge vibration monitoring. **Applied Sciences**, v. 12, n. 21, 2022.

SCHMIDT NICOLAS LE ROUX, F. B. M. Minimizing finite sums with the stochastic average gradient. **Mathematical Programming**, v. 162, p. 83–112, 2017.

SILVA, L. M. O. da. **Uma aplicação de árvores de decisão, redes neurais e KNN para a identificação de modelos ARMA não sazonais e sazonais**. 145 p. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2005.

SILVA, M. F. M. da. **Machine learning and computer vision techniques for damage detection and modal analysis**. 213 p. Tese (Doutorado) — Universidade Federal do Pará, Belém, 2020.

SNOEYS, R. et al. Trends in modal analysis. **Mechanical Systems and Signal Processing**, v. 1, n. 1, p. 5–27, 1985.

SOARES, M. et al. Using tranformation rules to align requirements and archictectural models. In: **Simpósio Brasileiro de Engenharia de Software**. Porto Alegre: Sociedade Brasileira de Computação, 2013. p. 26–35. 31 nov. de 2015. Disponível em: <http://cbsoft2013.unb.br/wp-content/uploads/2013/10/SBES-completo.pdf>.

SUN LI LIANG, M. L. X. L. S. Vibration-based damage detection in bridges via machine learning. p. 5123–5132, 2018.

A
Código Python 3 para gerar banco de dados de vibração no *Abaqus*

Python code

```
-*- coding: mbcs -*-
Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__
import csv
from csv import writer

def a1Geometry(L,pos):
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheet-
Size=200.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.Line(point1=(0.0, 0.0), point2=(L*pos/10, 0.0)) s.HorizontalConstraint(entity=g[2],
addUndoState=False)
p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=TWO_D_PLANAR,
type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-1']
p.BaseWire(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheet-
Size=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.Line(point1=(L*pos/10, 0.0), point2=(L*(pos+1)/10, 0.0))
s1.HorizontalConstraint(entity=g[2], addUndoState=False)
p = mdb.models['Model-1'].Part(name='Part-2', dimensionality=TWO_D_PLANAR,
type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-2']
```

```
p.BaseWire(sketch=s1)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-2']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
sheetSize=200.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.Line(point1=(L*(pos+1)/10, 0.0), point2=(L, 0.0))
s.HorizontalConstraint(entity=g[2], addUndoState=False)
p = mdb.models['Model-1'].Part(name='Part-3', dimensionality=TWO_D_PLANAR,
type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-3']
p.BaseWire(sketch=s)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-3']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']

      def a2Material(mat):
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
```

```
if mat==1:
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
engineeringFeatures=ON)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
referenceRepresentation=OFF)
mdb.models['Model-1'].Material(name='Material-1')
mdb.models['Model-1'].materials['Material-1'].Density(table=((2500, ), ))
mdb.models['Model-1'].materials['Material-1'].Elastic(table=((30000000.0,
0.2), ))
else:
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
engineeringFeatures=ON)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
referenceRepresentation=OFF)
mdb.models['Model-1'].Material(name='Material-1')
mdb.models['Model-1'].materials['Material-1'].Density(table=((7850,),))
mdb.models['Model-1'].materials['Material-1'].Elastic(table=((210000000.0,
0.3),))
def a3SectionProfs(h,base,damagesev):

mdb.models['Model-1'].RectangularProfile(name='Profile-1', a=base, b=h)
mdb.models['Model-1'].RectangularProfile(name='Profile-2',    a=base,    b=(1-
damagesev)*h)
mdb.models['Model-1'].BeamSection(name='Section-1',
integration=DURING_ANALYSIS, poissonRatio=0.0, profile='Profile-1',
material='Material-1', temperatureVar=LINEAR,
consistentMassMatrix=False)
mdb.models['Model-1'].BeamSection(name='Section-2',
integration=DURING_ANALYSIS, poissonRatio=0.0, profile='Profile-2',
material='Material-1', temperatureVar=LINEAR,
consistentMassMatrix=False)


def a4AssignSectOrient(L,pos):
PART 1 - ASSIGN SECTION
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-1']
e = p.edges
```

```
edges = e.getSequenceFromMask(mask=('[1 ]',), )
edges1 = e.getByBoundingBox(0,0,0,L*pos/10,0,0)
region = p.Set(edges=edges1, name='Set-1')
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
offsetType=MIDDLE_SURFACE, offsetField='',
thicknessAssignment=FROM_SECTION)
```

PART 2 - ASSIGN SECTION

```
p = mdb.models['Model-1'].parts['Part-2']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-2']
e = p.edges
edges = e.getSequenceFromMask(mask=('[1 ]',), )
edges2 = e.getByBoundingBox(L*pos/10.0,0,0,L*(pos+1)/10,0,0)
region = p.Set(edges=edges2, name='Set-2')
p = mdb.models['Model-1'].parts['Part-2']
p.SectionAssignment(region=region, sectionName='Section-2', offset=0.0,
offsetType=MIDDLE_SURFACE, offsetField='',
thicknessAssignment=FROM_SECTION)
```

PART 3 - ASSIGN SECTION

```
p = mdb.models['Model-1'].parts['Part-3']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-3']
e = p.edges
edges = e.getSequenceFromMask(mask=('[1 ]',), )
edges3 = e.getByBoundingBox(L*(pos+1)/10,0,0,L,0,0)
region = p.Set(edges=edges3, name='Set-3')
p = mdb.models['Model-1'].parts['Part-3']
p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
offsetType=MIDDLE_SURFACE, offsetField='',
thicknessAssignment=FROM_SECTION)
```

PART 1 - BEAM ORIENTATION

```
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-1']
e = p.edges
```

```
edges = e.getSequenceFromMask(mask=('[1 ]',), )
region = p.Set(edges=edges1, name='Set-4')
p = mdb.models['Model-1'].parts['Part-1']
p.assignBeamSectionOrientation(region=region,          method=N1_COSINES,
n1=(0.0, 0.0,
-1.0))

PART 2 - BEAM ORIENTATION
p = mdb.models['Model-1'].parts['Part-2']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-2']
e = p.edges
edges = e.getSequenceFromMask(mask=('[1 ]',), )
region = p.Set(edges=edges2, name='Set-5')
p = mdb.models['Model-1'].parts['Part-2']
p.assignBeamSectionOrientation(region=region,          method=N1_COSINES,
n1=(0.0, 0.0,
-1.0))

PART 3 BEAM ORIENTATION
p = mdb.models['Model-1'].parts['Part-3']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
p = mdb.models['Model-1'].parts['Part-3']
e = p.edges
edges = e.getSequenceFromMask(mask=('[1 ]',), )
region = p.Set(edges=edges3, name='Set-6')
p = mdb.models['Model-1'].parts['Part-3']
p.assignBeamSectionOrientation(region=region,          method=N1_COSINES,
n1=(0.0, 0.0,
-1.0))

      def a5Assembly():
a = mdb.models['Model-1'].rootAssembly
a.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-1']
a.Instance(name='Part-1-1', part=p, dependent=OFF)
p = mdb.models['Model-1'].parts['Part-2']
a.Instance(name='Part-2-1', part=p, dependent=OFF)
p = mdb.models['Model-1'].parts['Part-3']
```

```
a.Instance(name='Part-3-1', part=p, dependent=OFF)
session.viewports['Viewport: 1'].view.setValues(nearPlane=7230.47,
farPlane=8769.53, width=5437.32, height=2295.9, viewOffsetX=300.444,
viewOffsetY=-11.1008)
a = mdb.models['Model-1'].rootAssembly
a.InstanceFromBooleanMerge(name='Part-4', instances=(a.instances['Part-1-1'],
a.instances['Part-2-1'], a.instances['Part-3-1'],),
originalInstances=DELETE, domain=GEOMETRY)
a.makeIndependent(instances=(a.instances['Part-4-1'],))


def a7Step():
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
adaptiveMeshConstraints=ON)
mdb.models['Model-1'].FrequencyStep(name='Step-1', previous='Initial',
numEigen=6, acousticCoupling=AC_OFF)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Step-1')


def a8BC(L,bc):
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
predefinedFields=ON, connectors=ON,
adaptiveMeshConstraints=OFF)
a = mdb.models['Model-1'].rootAssembly
v1 = a.instances['Part-4-1'].vertices
verts1 = v1.getSequenceFromMask(mask=('[9 ]',), )
verts1 = v1.getByBoundingBox(0,0,0,1,0,0)
verts2 = v1.getByBoundingBox(L,0,0,L+1,0,0)
region = a.Set(vertices=(verts1,verts2), name='Set-8')
if bc ==1:
mdb.models['Model-1'].DisplacementBC(name='BC-1',  createStepName='Step-
1',
region=region, u1=0.0, u2=0.0, ur3=0.0, amplitude=UNSET, fixed=OFF,
distributionType=UNIFORM, fieldName='', localCsys=None)
else:
mdb.models['Model-1'].DisplacementBC(name='BC-1',  createStepName='Step-
1',
region=region, u1=0.0, u2=0.0, amplitude=UNSET, fixed=OFF,
distributionType=UNIFORM, fieldName='', localCsys=None)
```

```
def a9Mesh(L):
import mesh
session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON, loads=OFF,
bcs=OFF, predefinedFields=OFF, connectors=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
meshTechnique=ON)
minSizeFactor=0.1, constraint=FINER)

elemType1 = mesh.ElemType(elemCode=B22, elemLibrary=STANDARD)
a = mdb.models['Model-1'].rootAssembly
e1 = a.instances['Part-4-1'].edges
edges1 = e1.getByBoundingBox(0, 0, 0, L, 0, 0)
pickedRegions = (edges1,)
a.setElementType(regions=pickedRegions, elemTypes=(elemType1,))

a = mdb.models['Model-1'].rootAssembly
partInstances = (a.instances['Part-4-1'],)

a.generateMesh(regions=partInstances)
      def a11BeamOrient(L):
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
```

```
p = mdb.models['Model-1'].parts['Part-4']
e = p.edges
edges = e.getByBoundingBox(0, 0, 0, L, 0, 0)
region = p.Set(edges=edges, name='Set-9')
p = mdb.models['Model-1'].parts['Part-4']
p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
n1=(0.0, 0.0, -1.0))

def a12Job(jobname):
a = mdb.models['Model-1'].rootAssembly
a.regenerate()
name="Job-"
jobname=str(jobname)
finalname=name+jobname
print(finalname)
session.viewports['Viewport: 1'].setValues(displayedObject=a)
job1 = mdb.Job(name=finalname, model='Model-1', description='',
type=ANALYSIS,
atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
scratch='', resultsFormat=ODB)
job1.submit()
job1.waitForCompletion()
print("Processamento completo")

    def a13Path(L):
t = ((0, 0, 0),)
for i in range(1, 11):
t = t + ((i * L/10, 0, 0),)

session.Path(name='Path-1', type=POINT_LIST, expression=t)

    def a14XYData(jobname,mat,h,base,damsever,boundaries,pos):
import odbAccess

name="C:
SIMULIA
```

```
Macros-"
endname=".odb"
jobname=str(jobname)
finalname=name+jobname+endname
print(finalname)
      odb = session.openOdb(name = finalname)
o7 = session.odbs['C:/SIMULIA/Macros/Job-2.odb']
session.viewports['Viewport: 1'].setValues(displayedObject=odb)
session.viewports['Viewport:
1'].setValues(displayedObject=odb)
      session.XYDataFromHistory(name='EIGFREQ', odb=odb,
outputVariableName='Eigenfrequency: EIGFREQ for Whole Model', steps=(
'Step-1',), __linkedVpName__='Viewport: 1')
      1 MODO DE VIBRACAO
      session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=1)
session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, 'U2'))
pth = session.paths['Path-1']
session.XYDataFromPath(name='XYData-1', path=pth, includeIntersecti-
ons=True,
projectOntoMesh=False, pathStyle=PATH_POINTS, numIntervals=11,
projectionTolerance=0, shape=UNDEFORMED, labelType=TRUE_DISTANCE_X,
removeDuplicateXYPairs=True, includeAllElements=True)
      2 MODO DE VIBRACAO
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=2)
session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, 'U2'))
pth = session.paths['Path-1']
session.XYDataFromPath(name='XYData-2', path=pth, includeIntersecti-
ons=True,
projectOntoMesh=False, pathStyle=PATH_POINTS, numIntervals=11,
projectionTolerance=0, shape=UNDEFORMED, labelType=TRUE_DISTANCE_X,
removeDuplicateXYPairs=True, includeAllElements=True)
3 MODO DE VIBRACAO
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=3)
session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, 'U2'))
pth = session.paths['Path-1']
session.XYDataFromPath(name='XYData-3', path=pth, includeIntersecti-
```

```
ons=True,
projectOntoMesh=False, pathStyle=PATH_POINTS, numIntervals=11,
projectionTolerance=0, shape=UNDEFORMED, labelType=TRUE_DISTANCE_X,
removeDuplicateXYPairs=True, includeAllElements=True)
4 MODO DE VIBRACAO
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=5)
session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, 'U2'))
pth = session.paths['Path-1']
session.XYDataFromPath(name='XYData-4', path=pth, includeIntersecti-
ons=True,
projectOntoMesh=False, pathStyle=PATH_POINTS, numIntervals=11,
projectionTolerance=0, shape=UNDEFORMED, labelType=TRUE_DISTANCE_X,
removeDuplicateXYPairs=True, includeAllElements=True)
5 MODO DE VIBRACAO
session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0, frame=6)
session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, 'U2'))
pth = session.paths['Path-1']
session.XYDataFromPath(name='XYData-5', path=pth, includeIntersecti-
ons=True,
projectOntoMesh=False, pathStyle=PATH_POINTS, numIntervals=11,
projectionTolerance=0, shape=UNDEFORMED, labelType=TRUE_DISTANCE_X,
removeDuplicateXYPairs=True, includeAllElements=True)

x0 = session.xyDataObjects['EIGFREQ']
x1 = session.xyDataObjects['XYData-1']
x2 = session.xyDataObjects['XYData-2']
x3 = session.xyDataObjects['XYData-3']
x4 = session.xyDataObjects['XYData-4']
x5 = session.xyDataObjects['XYData-5']
    saveCSV('C:-beams3.csv' , x0,x1,x2,x3,x4,x5,mat,h,base,damsever,boundaries,pos)

odb.close()
def saveCSV(filename , x0,x1,x2,x3,x4,x5,mat,h,base,damsever,boundaries,pos):

with open(filename, 'a') as f:

linha=''
```

```
linha = linha+str(mat)+';'+str(h).replace('.',',')+';'+str(base).replace('.',',')+';'+str(damsever).replace('.'

for a in x0:
linha = linha + str(a[1]).replace('.' , ',') + ';'

for a in x1:
linha = linha + str(a[1]).replace('.' , ',') + ';'

for a in x2:
linha = linha + str(a[1]).replace('.' , ',') + ';'

for a in x3:
linha = linha + str(a[1]).replace('.' , ',') + ';'

for a in x4:
linha = linha + str(a[1]).replace('.' , ',') + ';'

for a in x5:
linha = linha + str(a[1]).replace('.' , ',') + ';'

linha = linha + ''
      f.write(linha)
      def deletetest(jobname):
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
```

```
import connectorBehavior
del session.xyDataObjects['EIGFREQ']
del session.xyDataObjects['XYData-1']
del session.xyDataObjects['XYData-2']
del session.xyDataObjects['XYData-3']
del session.xyDataObjects['XYData-4']
del session.xyDataObjects['XYData-5']
del session.paths['Path-1']
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
jobname=str(jobname)
name='Job-'
finalname=name+jobname
del mdb.jobs[finalname]
session.viewports['Viewport:
1'].assemblyDisplay.setValues(loads=ON, bcs=ON,
predefinedFields=ON, connectors=ON)
del mdb.models['Model-1'].boundaryConditions['BC-1']
session.viewports['Viewport: 1'].assemblyDisplay.setValues(loads=OFF, bcs=OFF,
predefinedFields=OFF, connectors=OFF, adaptiveMeshConstraints=ON)
del mdb.models['Model-1'].steps['Step-1']
session.viewports['Viewport: 1'].assemblyDisplay.setValues(step='Initial')
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
adaptiveMeshConstraints=OFF)
a = mdb.models['Model-1'].rootAssembly
del a.features['Part-4-1']
p1 = mdb.models['Model-1'].parts['Part-3']
session.viewports['Viewport: 1'].setValues(displayedObject=p1)
del mdb.models['Model-1'].profiles['Profile-1']
del mdb.models['Model-1'].profiles['Profile-2']
del mdb.models['Model-1'].sections['Section-1']
del mdb.models['Model-1'].sections['Section-2']
del mdb.models['Model-1'].materials['Material-1']
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=OFF,
engineeringFeatures=OFF)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
referenceRepresentation=ON)
p1 = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p1)
```

```
del mdb.models['Model-1'].parts['Part-1']
del mdb.models['Model-1'].parts['Part-2']
del mdb.models['Model-1'].parts['Part-3']
del mdb.models['Model-1'].parts['Part-4']
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
del mdb.models['Model-1'].rootAssembly.sets['Set-8']
a = mdb.models['Model-1'].rootAssembly
del a.features['Datum csys-1']

    def Teste(L,mat,h,base,damagesev,bc,jobname,pos):
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior

a1Geometry(L,pos)
a2Material(mat)
a3SectionProfs(h,base,damagesev)
a4AssignSectOrient(L,pos)
a5Assembly()
a7Step()
a8BC(L,bc)
a9Mesh(L)
a11BeamOrient(L)
a12Job(jobname)
```

```
a13Path(L)
print('2')
a14XYData(jobname,mat,h,base,damagesev,bc,pos)
deletetest(jobname)
def case1Mestrado():

count = 1
for i in range(16,18):
for mat in range(1,2):
for height in range (1,4):
for bs in range (1,3):
for damage in range (1,5):
for boundaries in range (1,3):
for position in range (1,5):
if damage==1 and position >1:
break
lenght=4000+(i-1)*250
h=(4000+(i-1)*250)*(0.1-0.01*height)
base=(4000+(i-1)*250)*(0.1-0.01*height)/(bs+1)
damsever=(damage-1)*0.1
jobname=str(count)
Teste(lenght,mat,h,base,damsever,boundaries,jobname,position)
count=count+1
counts=str(count)
print(counts)
```

B
Código Python 3 de pré-processamento e treinamento de modelos de *Machine Learning*

```
        df.corr()
.pipe(
lambda df1: pd.DataFrame(
np.tril(df1, k=-1),
columns=df.columns,
index=df.columns,
)
)
.stack()
.rename("kendall")
.pipe(
lambda s: s[
s.abs() > threshold
].reset_index()
)
.query("level_0 not in level_1")
)

        In[887]:

        def plot_confusion_matrix(y_train, y_test, X_train, X_test, model):
realiza a previsão de treino e teste
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
calcula a acurácia de treino e teste
acc_train = accuracy_score(y_pred_train, y_train)
acc_test = accuracy_score(y_pred_test, y_test)
calcula a matriz de confusao de treino e teste
cm_train = confusion_matrix(y_train, y_pred_train)
cm_test = confusion_matrix(y_test, y_pred_test)
normaliza as matrizes de confusao
cm_norm_train = cm_train / cm_train.astype(float).sum(axis=1)[:, np.newaxis]
cm_norm_test = cm_test / cm_test.astype(float).sum(axis=1)[:, np.newaxis]
cria um dataframe com as matrizes de confusao
```

```
cm_train_df = pd.DataFrame(cm_train,
index = ['0','1'],
columns = ['0','1'])
cm_test_df = pd.DataFrame(cm_test,
index = ['0','1'],
columns = ['0','1'])
cm_norm_train_df = pd.DataFrame(cm_norm_train,
index = ['0','1'],
!/usr/bin/env python
coding: utf-8
    Frequency - B22
    1. Importing packages and functions
    1.1 Packages
    In[884]:
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_validate
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import recall_score, accuracy_score,
precision_score,f1_score
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.model_selection import GridSearchCV,
StratifiedKFold, KFold, cross_val_score,
cross_val_predict, ShuffleSplit
from sklearn.model_selection import RandomizedSearchCV
from sklearn import model_selection
from IPython.display import display
import warnings
import datetime
      1.2 Parameters
      In[885]:
      pd.options.display.max_columns = None
pd.options.display.max_rows = None
sns.set_palette("tab10")
sns.set_style("whitegrid")
seed = 42
      In[886]:
      def correlated_columns(df, threshold):
return (
columns = ['0','1'])
cm_norm_test_df = pd.DataFrame(cm_norm_test,
index = ['0','1'],
columns = ['0','1'])
inicío do plot
plt.figure(figsize=(10, 10))

subplot 1 (treino relativo)
plt.subplot(2, 2, 1)
sns.heatmap(cm_norm_train_df, annot=True,cmap='Blues', cbar = 0)
plt.title('Train Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label=:0.4f'.format(acc_train))

subplot 2 (teste relativo)
plt.subplot(2, 2, 2)
sns.heatmap(cm_norm_test_df, annot=True,cmap='Blues', cbar = 0)
plt.title('Test Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label=:0.4f'.format(acc_test))
      subplot 1 (treino absoluto)
```

```
plt.subplot(2, 2, 3)
sns.heatmap(cm_train_df, annot=True,cmap='Blues', cbar = 0)
plt.title('Train Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label=:0.4f'.format(acc_train))
      subplot 2 (teste absoluto)
plt.subplot(2, 2, 4)
sns.heatmap(cm_test_df, annot=True,cmap='Blues', cbar = 0)
plt.title('Test Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label=:0.4f'.format(acc_test))
      plt.tight_layout() plt.show()
      In[888]:
      cria um gráfico com as matrizes de confusão de treino e de teste
def cf_plot(clf, X_train, X_test, y_train, y_test, method):
acc_train = clf.score(X_train, y_train)
acc_test = clf.score(X_test, y_test)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
cf_matrix_train = confusion_matrix(y_train, y_pred_train, normalize='true')
cf_matrix_test = confusion_matrix(y_test, y_pred_test, normalize='true')
disp_cf_matrix_train = ConfusionMatrixDisplay(
cf_matrix_train)
disp_cf_matrix_test = ConfusionMatrixDisplay(
cf_matrix_test)
f, axes = plt.subplots(1, 2, figsize=(9, 6), sharey='row')
axes[0].grid(False)
axes[1].grid(False)
disp_cf_matrix_train.plot(ax=axes[0], cmap='Blues')
disp_cf_matrix_train.ax_.set_xlabel(
'Predicted label=:0.3f'.format(acc_train)
)
disp_cf_matrix_test.plot(ax=axes[1], cmap='Blues')
disp_cf_matrix_test.ax_.set_xlabel(
'Predicted label=:0.3f'.format(acc_test)
)
title_train = method+' (Train)'
title_test = method+' (Test)'
disp_cf_matrix_train.ax_.set_title(title_train)
```

```
disp_cf_matrix_test.ax_.set_title(title_test)
disp_cf_matrix_train.im_.colorbar.remove()
disp_cf_matrix_test.im_.colorbar.remove()
plt.show()
    In[952]:
    def models_score(models, X_train, y_train, X_test, y_test, scoring =
'recall', n_splits=10, figsize = (20, 12)):
results_cv = []
results = []
names = []
best_rec_test = 0
best_clf = ''
for name, model in models:
names.append(name)
cross validation
skfold = model_selection.StratifiedKFold(n_splits=n_splits)
cv_results = model_selection.cross_val_score(model, X_train, y_train,
cv=skfold, scoring=scoring)
results_cv.append(cv_results)  rec cv
model training
time1 = datetime.datetime.now()
model.fit(X_train, y_train)
time2 = datetime.datetime.now()
model predict
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
accuracy
train_acc = round(model.score(X_train,y_train),3)
test_acc = round(model.score(X_test,y_test),3)
confusion matrix
tn_train, fp_train, fn_train, tp_train = confusion_matrix(y_train,
y_pred_train).ravel()
tn_test, fp_test, fn_test, tp_test = confusion_matrix(y_test, y_pred_test).ravel()
precision
train_prec=round(precision_score(y_train, y_pred_train),3)
test_prec=round(precision_score(y_test, y_pred_test),3)
recall
train_rec=round(recall_score(y_train, y_pred_train),3)
test_rec=round(recall_score(y_test, y_pred_test),3)
```

```
if test_rec > best_rec_test:
best_rec_test = round(test_rec,3)
best_clf = model
best_name = name
f1 score
train_f1=round(f1_score(y_train, y_pred_train),3)
test_f1=round(f1_score(y_test, y_pred_test),3)
training time
duration=time2-time1
df
results.append('Name': name,
'Recall (train)': round(train_rec,3),
'Recall (test)': round(test_rec,3),
'Acc (train)': round(train_acc,3),
'Acc (test)': round(test_acc,3),
'Precision (train)': round(train_prec,3),
'Precision (test)': round(test_prec,3),
'F1 Score (train)': round(train_f1,3),
'F1 Score (test)': round(test_f1,3),
'CV Recall (train)': round(np.mean(cv_results),3))
'Training time (s)': duration.total_seconds())
results_df = pd.DataFrame(results).sort_values(by='Recall (test)', ascending=False).reset_index(drop = True)
results_cv_df = pd.DataFrame(results_cv).T
results_cv_df.columns = names
print('————————-Results———————— ')
print('Best recall in train (CV mean): :0.3f'.format(np.max(np.mean(results_cv, axis=1))))
print('Best recall in test: :0.3f'.format(results_df['Recall (test)'].max()))
print('Best precision in test: :0.3f'.format(results_df['Precision (test)'].max()))
print('Best accuracy in test: :0.3f'.format(results_df['Acc (test)'].max()))
print('Best F1 score in test: :0.3f'.format(results_df['F1 Score (test)'].max()))
print('————————-Best confusion matrix (Recall test)———————— ')
cf_plot(best_clf, X_train, X_test, y_train, y_test, best_name)
print('————————Models comparison———————— ')
display(results_df)
print(results_df.to_latex(index=False))
print('————————-Visualization———————— ')
comparison_visualization(results_cv_df, results_df, figsize)
```

```
return results_cv_df, results_df
    In[950]:
    def comparison_visualization(results_cv_df, results_test, figsize = (20,
12)):
fig, axs = plt.subplots(2,1,figsize=figsize)
axs = axs.flatten()
sns.barplot(x='Name',y = 'Recall (test)', data = results_test, ax=axs[0]).set(ylabel='Recall')
axs[0].title.set_text('Test recall results')
sns.barplot(x='Name',y = 'Acc (test)', data =
results_test, ax=axs[1]).set(ylabel='Acc')
axs[1].title.set_text('Test accuracy results')
sns.barplot(x='Name',y = 'Precision (test)', data = results_test,
ax=axs[2]).set(ylabel='Precision')
axs[2].title.set_text('Test precision results')
sns.barplot(x='Name',y = 'F1 Score (test)', data = results_test,
ax=axs[3]).set(ylabel='F1 Score')
axs[3].title.set_text('Test F1 score results')
sns.boxplot(data=results_cv_df.reindex(columns=list(results_test['Name'])),
ax=axs[4]).set(xlabel='Algorithm', ylabel='Avg accuracy')
axs[1].title.set_text('Train CV accuracy results')
plt.setp(axs, ylim=(0,1))
    labels = results_test['Name'].values
recall = results_test['Recall (test)'].values
accuracy = results_test['Acc (test)'].values
precision = results_test['Precision (test)'].values
f1_score = results_test['F1 Score (test)'].values
    x = np.arange(len(labels))
width = 0.15
    fig, axs = plt.subplots(2,1,figsize=figsize,sharex=True)
rects1=axs[0].bar(x-2*width,recall,width,label='Recall (test)')
rects2=axs[0].bar(x-3*width/4,accuracy,width,label='Accuracy (test)')
rects3=axs[0].bar(x+3*width/4,precision,width,label='Precision (test)')
rects4=axs[0].bar(x+2*width,f1_score,width,label='F1 Score (test)')
    axs[0].set_ylabel('Scores')
axs[0].set_xticks(x)
axs[0].set_xticklabels(labels)
axs[0].legend()
    axs[0].bar_label(rects1, padding=3)
axs[0].bar_label(rects2, padding=3)
```

```
axs[0].bar_label(rects3, padding=3)
axs[0].bar_label(rects4, padding=3)
    fig.tight_layout()
    sns.boxplot(data=results_cv_df.reindex(columns=list(results_test['Name'])),
ax=axs[1],palette='Blues_r').set(xlabel='Algorithm', ylabel='Avg recall')
axs[1].title.set_text('Train CV recall results')
plt.setp(axs[0], ylim=(0,1.3))
plt.show()
```

2. Database

2.1 Importing database as pandas dataframe

In[891]:

```
data = pd.read_csv('db-beams-clean-freq-v2.csv',sep=';',low_memory=False,decimal=",")
```

In[892]:

data.head() exibindo as primeiras linhas

In[893]:

data.tail() exibindo as últimas linhas

In[894]:

```
print('A base possui', data.shape[0], 'linhas (vigas) e', data.shape[1], 'variá-
veis (atributos)')
```

In[895]:

```
sns.pairplot(data[['Lenght','I','Freq 1','Freq 2','Freq 3','Freq 4','Freq 5','Tar-
get']],hue='Target')
```

2.2 Data types

In[896]:

```
data.info(verbose=True)
```

3. Brief data analysis

In[897]:

```
df_zeros = (data == 0).mean()
df_zeros = df_zeros[df_zeros > 0] * 100
print("Colunas com valores zero (qtd relativa): ".format(df_zeros.sort_values(ascending
= False)))
df_1 = data[data['Target'] == 1]
df_0 = data[data['Target'] == 0]
```

3.1.1.1 Numerical variables per target

In[898]:

```
df_1.drop(labels=['Beam Id'], axis=1).describe().apply(lambda s:
s.apply('0:.2f'.format)).T
```

In[899]:

df_0.drop(labels=['Beam Id'], axis=1).describe().apply(lambda s: s.apply('0:.2f'.format)).T

3.1.3 Target (damaged beams)

In[901]:

data['Target_hum'] = data.Target.map(0: 'Not damaged', 1: 'Damaged')
ax = sns.countplot(data=data, x='Target_hum', alpha=0.7,order=['Not damaged','Damaged'])
plt.title('Beams', fontsize=16)
plt.ylabel(' beams', fontsize=12)
plt.xlabel('Damage?', fontsize=12)
ax.ticklabel_format(style='plain', axis='y')

for p in ax.patches:
ax.annotate('0:,'.format(p.get_height()), (p.get_x()+0.4, (p.get_height()/2)), ha='center', va='center', color='white', size=16)
plt.show()

print('
data.Target_hum.value_counts()/data.shape[0] * 100

3.2 Analysis of input data

In[902]:

const_columns = list(data.columns[data.nunique(dropna=False) <= 1])
print(const_columns)
all_columns = list(data.columns)
print(all_columns)
used_columns = [x for x in all_columns if (x not in const_columns)]
print(used_columns)

4. Train and test databases split

4.1 Split and feature scaling

In[998]:

divide a base em partições de treino (70%) e teste (30%) com dados estratificados pela variável alvo
X_train, X_test, y_train, y_test = train_test_split(
data[used_columns].drop(['Beam Id','Target_hum','Target'], axis=1),
data['Target'],
test_size=0.3,
random_state=42,
stratify=data['Target'])

cols=X_train.columns
scaler = StandardScaler()

```
X_test_sc=scaler.fit_transform(X_test)
X_train_sc=scaler.fit_transform(X_train)
    data_train = pd.DataFrame(X_train_sc,columns=cols)
data_test = pd.DataFrame(X_test_sc,columns=cols)
data_train['Target']=list(y_train)
data_test['Target']=list(y_test)
pos_train=data_train['Position']
dam_train=data_train['Damage %']
pos_test=data_test['Position']
dam_test=data_test['Damage %']
    print("Número de registros em X_train: ", X_train_sc.shape)
print("Número de registros em y_train: ", y_train.shape)
print("Número de registros em X_test: ", X_test_sc.shape)
print("Número de registros em y_test: ", y_test.shape)
    df_train1 = data_train[data_train['Target'] == 1]
df_train0 = data_train[data_train['Target'] == 0]
print(,df_train1['Target'].size)
print(,df_train0['Target'].size)
    undersample = RandomUnderSampler(sampling_strategy='majority', ran-
dom_state = 42)
X_under, y_under = undersample.fit_resample(X_train, y_train)
X_under_sc=scaler.fit_transform(X_under)
data_under = pd.DataFrame(X_under_sc,columns=cols)
data_under['Target']=list(y_under)
pos_under=data_under['Position']
dam_under=data_under['Damage %']
df_under1 = data_under[data_under['Target'] == 1]
df_under0 = data_under[data_under['Target'] == 0]
print(,data_under['Target'].size)
print(,df_under0['Target'].size)
    smote = SMOTE(random_state = 42)
X_smote, y_smote = smote.fit_resample(X_train, y_train)
X_smote_sc=scaler.fit_transform(X_smote)
data_smote = pd.DataFrame(X_smote_sc,columns=cols)
data_smote['Target']=list(y_smote)
pos_smote=data_smote['Position']
dam_smote=data_smote['Damage %']
df_smote1 = data_smote[data_smote['Target'] == 1]
df_smote0 = data_smote[data_smote['Target'] == 0]
```

```
print(,data_smote['Target'].size)
print(,df_smote0['Target'].size)
```

4.2 Samples

In[904]:

```
data_train['Target_hum'] = data_train.Target.map(0: 'Not damaged', 1: 'Damaged')
ax = sns.countplot(data=data_train, x='Target_hum', alpha=0.7,order=['Not damaged','Damaged'])
plt.title('Imbalanced sampling', fontsize=16)
plt.ylabel(' beams', fontsize=12)
plt.xlabel('Damaged?', fontsize=12)
ax.ticklabel_format(style='plain', axis='y')
```

for p in ax.patches:

```
ax.annotate('0:,'.format(p.get_height()), (p.get_x()+0.4, (p.get_height()/2)), ha='center', va='center', color='white', size=16)
plt.show()

print('
data_train.Target_hum.value_counts()/data_train.shape[0] * 100
```

In[905]:

```
sns.pairplot(data_train[['Lenght','I','Freq 1','Freq 2','Freq 3','Freq 4','Freq 5','Target']],hue='Target')
```

In[906]:

```
data_under['Target_hum'] = data_under.Target.map(0: 'Not damaged', 1: 'Damaged')
ax = sns.countplot(data=data_under, x='Target_hum', alpha=0.7,order=['Not damaged','Damaged'])
plt.title('Undersampling', fontsize=16)
plt.ylabel(' beams', fontsize=12)
plt.xlabel('Damaged?', fontsize=12)
ax.ticklabel_format(style='plain', axis='y')
```

for p in ax.patches:

```
ax.annotate('0:,'.format(p.get_height()), (p.get_x()+0.4, (p.get_height()/2)), ha='center', va='center', color='white', size=16)
plt.show()
```

print('

```
data_under.Target_hum.value_counts()/data_under.shape[0] * 100
```

In[907]:

```
sns.pairplot(data_under[['Lenght','I','Freq 1','Freq 2','Freq 3','Freq 4','Freq
5','Target']],hue='Target')
In[908]:
data_smote['Target_hum'] = data_smote.Target.map(0: 'Not damaged',
1: 'Damaged')
ax = sns.countplot(data=data_smote, x='Target_hum', alpha=0.7,order=['Not
damaged','Damaged'])
plt.title('Oversampling', fontsize=16)
plt.ylabel(' beams', fontsize=12)
plt.xlabel('Damaged?', fontsize=12)
ax.ticklabel_format(style='plain', axis='y')

for p in ax.patches:
ax.annotate('0:,'.format(p.get_height()), (p.get_x()+0.4, (p.get_height()/2)),
ha='center', va='center', color='white', size=16)
plt.show()
print('data_smote.Target_hum.value_counts()/data_smote.shape[0] * 100
In[909]:
sns.pairplot(data_smote[['Lenght','I','Freq 1','Freq 2','Freq 3','Freq 4','Freq
5','Target']],hue='Target')

5. Pre-analysis
5.1 Target correlation analysis
In[910]:
df_corr = data_train.corr(method='kendall')['Target'][:]
pd.DataFrame(df_corr).sort_values(by="Target", ascending = False)
5.2 Multicolinearity
In[911]:
corr_data_train=data_train.drop(['Target_hum'],axis=1)
correlated_columns(corr_data_train,0.6)
In[912]:
data['Target_hum'] = data.Target.map(0: 'Not damaged', 1: 'Damaged')
ax = sns.countplot(data=data, x='Target_hum', alpha=0.7)
plt.title('Beams', fontsize=16)
plt.ylabel(' beams', fontsize=12)
plt.xlabel('Damaged?', fontsize=12)
ax.ticklabel_format(style='plain',
axis='y')
```

```
        for p in ax.patches:
ax.annotate('0:,'.format(p.get_height()),  (p.get_x()+0.4,  (p.get_height()/2)),
ha='center', va='center', color='white', size=16)
plt.show()
        print('
data.Target_hum.value_counts()/data.shape[0] * 100
        X_train.drop(labels=['Position'], axis = 1, inplace = True)
X_test.drop(labels=['Position'], axis = 1, inplace = True)
X_under.drop(labels=['Position'], axis = 1, inplace = True)
X_smote.drop(labels=['Position'], axis = 1, inplace = True)
        X_train.drop(labels=['Damage
X_test.drop(labels=['Damage
X_under.drop(labels=['Damage
X_smote.drop(labels=['Damage
        6. Models
        6.1 Linear regression
        In[953]:
        models = []
models.append(('LINr 01', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.1)))
models.append(('LINr 02', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.01)))
models.append(('LINr 03', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.001)))
models.append(('LINr 04', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.0001)))
models.append(('LINr 05', RidgeClassifier(max_iter=2000, random_state=seed,solver='sag',tol=0.1)))
models.append(('LINr 06', RidgeClassifier(max_iter=2000, random_state=seed,solver='sag',tol=0.01)))
models.append(('LINr 07', RidgeClassifier(max_iter=2000, random_state=seed,solver='sag',tol=0.001)))
models.append(('LINr 08', RidgeClassifier(max_iter=2000, random_state=seed,solver='sag',tol=0.0001)))
        6.1.1 Imbalanced
        In[954]:
        a,b = models_score(models, X_train, y_train, X_test, y_test, scoring =
'recall', n_splits=10)
        6.1.2 Under-sampling
        In[957]:
        a,b = models_score(models, X_under, y_under, X_test, y_test, scoring =
'recall', n_splits=10)
        6.1.3 Over-sampling
        In[958]:
        a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring =
'recall', n_splits=10)
        6.2 Logistic regression
```

In[967]:

models = []

models.append(('LOGr 01', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.01)))

models.append(('LOGr 02', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.001)))

models.append(('LOGr 03', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.0001)))

models.append(('LOGr 04', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.00001)))

models.append(('LOGr 05', LogisticRegression(random_state=seed, max_iter=2000,solver='lbfgs',tol=0.01)))

models.append(('LOGr 06', LogisticRegression(random_state=seed, max_iter=2000,solver='lbfgs',tol=0.001)))

models.append(('LOGr 07', LogisticRegression(random_state=seed, max_iter=2000,solver='lbfgs',tol=0.0001)))

models.append(('LOGr 08', LogisticRegression(random_state=seed, max_iter=2000,solver='lbfgs',tol=0.00001)))

6.2.1 Imbalanced

In[968]:

a,b = models_score(models, X_train, y_train, X_test, y_test, scoring = 'recall', n_splits=10)

6.2.2 Under-sampling

In[969]:

a,b = models_score(models, X_under, y_under, X_test, y_test, scoring = 'recall', n_splits=10)

6.2.3 Over-sampling

In[970]:

a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring = 'recall', n_splits=10)

6.3 Decision Tree

In[971]:

models = []

models.append(('DT 05', DecisionTreeClassifier(random_state=seed, max_depth = 5)))

models.append(('DT 10', DecisionTreeClassifier(random_state=seed, max_depth = 10)))

models.append(('DT 15', DecisionTreeClassifier(random_state=seed, max_depth = 15)))

```
models.append(('DT 20', DecisionTreeClassifier(random_state=seed, max_depth
= 20)))
models.append(('DT 25', DecisionTreeClassifier(random_state=seed, max_depth
= 25)))
models.append(('DT 30', DecisionTreeClassifier(random_state=seed, max_depth
= 30)))
models.append(('DT 35', DecisionTreeClassifier(random_state=seed, max_depth
= 35)))
models.append(('DT 40', DecisionTreeClassifier(random_state=seed, max_depth
= 40)))
```

6.3.1 Imbalanced sampling

In[972]:

```
a,b = models_score(models, X_train, y_train, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.3.2 Under-sampling

In[973]:

```
a,b = models_score(models, X_under, y_under, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.3.3 Over-sampling

In[974]:

```
a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.4 Random Forest

In[975]:

```
models = []
models.append(('RF 05', RandomForestClassifier(random_state=seed,
max_depth = 5, n_jobs=4)))
models.append(('RF 10', RandomForestClassifier(random_state=seed,
max_depth = 10, n_jobs=4)))
models.append(('RF 15', RandomForestClassifier(random_state=seed,
max_depth = 15, n_jobs=4)))
models.append(('RF 20', RandomForestClassifier(random_state=seed,
max_depth = 20, n_jobs=4)))
models.append(('RF 25', RandomForestClassifier(random_state=seed,
max_depth = 25, n_jobs=4)))
models.append(('RF 30', RandomForestClassifier(random_state=seed,
max_depth = 30, n_jobs=4)))
models.append(('RF 35', RandomForestClassifier(random_state=seed,
max_depth = 35, n_jobs=4)))
```

models.append(('RF 40', RandomForestClassifier(random_state=seed, max_depth = 40, n_jobs=4)))

6.4.1 Imbalanced sampling

In[976]:

a,b = models_score(models, X_train, y_train, X_test, y_test, scoring = 'recall', n_splits=10)

6.4.2 Under-sampling

In[977]:

a,b = models_score(models, X_under, y_under, X_test, y_test, scoring = 'recall', n_splits=10)

6.4.3 Over-sampling

In[978]:

a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring = 'recall', n_splits=10)

6.5 Support Vector Machines

In[979]:

models = []

models.append(('SVM 01',SVC(random_state=seed,kernel='linear',tol=0.01,class_weight='balanced')))
models.append(('SVM 02',SVC(random_state=seed,kernel='linear',tol=0.001,class_weight='balanced')))
models.append(('SVM 03',SVC(random_state=seed,kernel='poly',tol=0.01,class_weight='balanced')))
models.append(('SVM 04',SVC(random_state=seed,kernel='poly',tol=0.001,class_weight='balanced')))
models.append(('SVM 05',SVC(random_state=seed,kernel='rbf',tol=0.01,class_weight='balanced')))
models.append(('SVM 06',SVC(random_state=seed,kernel='rbf',tol=0.001,class_weight='balanced')))
models.append(('SVM 07',SVC(random_state=seed,kernel='sigmoid',tol=0.01,class_weight='balanced')
models.append(('SVM 08',SVC(random_state=seed,kernel='sigmoid',tol=0.001,class_weight='balanced

6.5.1 Imbalanced sampling

In[980]:

a,b = models_score(models,X_train, y_train, X_test, y_test, scoring = 'recall', n_splits=10)

6.5.2 Under-sampling

In[981]:

a,b = models_score(models,X_under, y_under, X_test, y_test, scoring = 'recall', n_splits=10)

6.5.3 Over-sampling

In[982]:

a,b = models_score(models,X_smote, y_smote, X_test, y_test, scoring = 'recall', n_splits=10)

6.6 Neural Network (Multi-layer perceptron)

In[983]:

models = []
models.append(('MLP 01', MLPClassifier(random_state=seed, hidden_layer_sizes=(5))))
models.append(('MLP 02', MLPClassifier(random_state=seed, hidden_layer_sizes=(5,5))))
models.append(('MLP 03', MLPClassifier(random_state=seed, hidden_layer_sizes=(10))))
models.append(('MLP 04', MLPClassifier(random_state=seed, hidden_layer_sizes=(10,10))))
models.append(('MLP 05', MLPClassifier(random_state=seed, hidden_layer_sizes=(15))))
models.append(('MLP 06', MLPClassifier(random_state=seed, hidden_layer_sizes=(15,15))))
models.append(('MLP 07', MLPClassifier(random_state=seed, hidden_layer_sizes=(20))))
models.append(('MLP 08', MLPClassifier(random_state=seed, hidden_layer_sizes=(20,20))))

6.6.1 Imbalanced sampling

In[984]:

a,b = models_score(models, X_train, y_train, X_test, y_test, scoring = 'recall', n_splits=10)

6.6.2 Undersampling

In[985]:

a,b = models_score(models, X_under, y_under, X_test, y_test, scoring = 'recall', n_splits=10)

6.6.3 Oversampling

In[986]:

a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring = 'recall', n_splits=10)

6.7 K-nearest neighbor

In[ ]:

In[987]:

models = []
models.append(('KNN 01', KNeighborsClassifier(n_neighbors=1,weights='distance')))
models.append(('KNN 02', KNeighborsClassifier(n_neighbors=3,weights='distance')))
models.append(('KNN 03', KNeighborsClassifier(n_neighbors=5,weights='distance')))
models.append(('KNN 04', KNeighborsClassifier(n_neighbors=15,weights='distance')))
models.append(('KNN 05', KNeighborsClassifier(n_neighbors=31,weights='distance')))
models.append(('KNN 06', KNeighborsClassifier(n_neighbors=51,weights='distance')))

```
models.append(('KNN 07', KNeighborsClassifier(n_neighbors=101,weights='distance')))
models.append(('KNN 08', KNeighborsClassifier(n_neighbors=151,weights='distance')))
```

6.7.1 Imbalanced sampling

In[988]:

```
a,b = models_score(models,X_train, y_train, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.7.2 Under-sampling

In[989]:

```
a,b = models_score(models,X_under, y_under, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.7.3 Over-sampling

In[990]:

```
a,b = models_score(models,X_smote, y_smote, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.8 Multi-models analysis

In[991]:

```
models = []
models.append(('LINr', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.1)))
models.append(('LOGr', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.01)
models.append(('DT', DecisionTreeClassifier(random_state=seed, max_depth =
5)))
models.append(('RF', RandomForestClassifier(random_state=seed, max_depth
= 30, n_jobs=4)))
models.append(('SVM',SVC(random_state=seed,kernel='linear',tol=0.01,class_weight='balanced')))
models.append(('MLP', MLPClassifier(random_state=seed, hidden_layer_sizes=(5))))
models.append(('KNN',KNeighborsClassifier(n_neighbors=5,weights='distance')))
```

6.8.1 Imbalanced sampling

In[992]:

```
a,b = models_score(models, X_train, y_train, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.8.2 Undersampling

In[993]:

```
models = []
models.append(('LINr', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.01)))
models.append(('LOGr', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.01)
models.append(('DT', DecisionTreeClassifier(random_state=seed, max_depth =
5)))
models.append(('RF', RandomForestClassifier(random_state=seed, n_jobs=-1,
max_depth = 10)))
```

```
models.append(('SVM',SVC(random_state=seed,kernel='poly',tol=0.01,class_weight='balanced')))
models.append(('MLP', MLPClassifier(random_state=seed, hidden_layer_sizes=(10,10))))
models.append(('KNN',KNeighborsClassifier(n_neighbors=1,weights='distance')))
```

In[994]:

```
a,b = models_score(models, X_under, y_under, X_test, y_test, scoring =
'recall', n_splits=10)
```

6.8.3 Oversampling

In[995]:

```
models = []
models.append(('LINr', RidgeClassifier(max_iter=2000, random_state=seed,solver='lsqr',tol=0.01)))
models.append(('LOGr', LogisticRegression(random_state=seed, max_iter=2000,solver='saga',tol=0.01)
models.append(('DT', DecisionTreeClassifier(random_state=seed, max_depth =
35)))
models.append(('RF', RandomForestClassifier(random_state=seed, n_jobs=-1,
max_depth = 30)))
models.append(('SVM',SVC(random_state=seed,kernel='linear',tol=0.01,class_weight='balanced')))
models.append(('MLP', MLPClassifier(random_state=seed, hidden_layer_sizes=(5,5))))
models.append(('KNN',KNeighborsClassifier(n_neighbors=15,weights='distance')))
```

In[996]:

```
a,b = models_score(models, X_smote, y_smote, X_test, y_test, scoring =
'recall', n_splits=10)
```