UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE GEOCIÊNCIAS

MAPEAMENTO DE RISCO EM SÃO SEBASTIÃO E BERTIOGA UTILIZANDO FERRAMENTAS DE MACHINE LEARNING

TIAGO SATOSHI OBARA

Orientador: Prof. Dr.Vinicius Hector Abud Louro
Coorientador: Tiago Antonelli

SÃO PAULO
2023

Resumo

Movimentos gravitacionais de massa são problemas recorrentes no litoral paulista causando danos sociais e econômicos, esses eventos estão associados com vários fatores como a geologia, pedologia, topografia, clima e a vegetação. Como forma de apoio às instituições de monitoramento e remediação desses danos, como a defesa civil, este trabalho produziu um mapa de susceptibilidade das regiões de Bertioga e São Sebastião baseado em aprendizado de máquina (*machine learning*). Para tal foram utilizados dados de geologia, pedologia, profundidade do solo, topografia, direção de inclinação de encosta, ângulo de inclinação da encosta, vegetação e pluviometria acumulada de 72 horas. O aprendizado de máquina foi aplicado com o método de *perceptron* multicamadas para realizar a classificação dos dados e apresentar um mapa de probabilidade de movimentação. O modelo classificou a área de estudo com probabilidades de deslizamentos chegando a 76% e apresentou uma boa validação com um f1-score acima de 0,90, mas com uma base de dados mais completa o modelo pode ser mais acurado.

Abstract

Landslides are a recurring problem on the coast of São Paulo, causing social and economic damage. These events are associated with several factors such as geology, pedology, topography, climate and vegetation. As a way of supporting institutions that monitors and remediates this damage, such as Defesa Civil, this work produced a landslide susceptibility map of the regions of Bertioga and São Sebastião based on machine learning. To this end, data on geology, pedology, soil depth, topography, slope, aspect, vegetation and 72-hour accumulated rainfall were used. Machine learning was applied with the multilayer perception method to perform data classification and present a landslide probability map. The model classified the study area with landslide probabilities reaching 76% and presented good validation with an f1-score above 0.90, but with a more complete database the model can be more accurate.

SUMÁRIO

1. INTRODUÇÃO

Movimentos Gravitacionais de Massa (MGMs) são fenômenos naturais que atuam para reequilibrar a encosta, por exemplo um bloco rochoso deslizando para baixo num plano inclinado. Porém, nos MGMs esse "bloco" é constituído por materiais como solo, rocha e vegetação que quando atingem uma infraestrutura ou uma pessoa podem causar danos, prejuízos e, inclusive, óbitos. Segundo Macedo e Sandre (2022), no Brasil, de 1988 a junho 2022 houveram 4146 fatalidades.

Para prevenir de tais acidentes, trabalhos como o de Pimentel e Santos (2018) e Bitar (2014) elaboram métodos de mapear as zonas de risco. Em geral, esses trabalhos focam na elaboração do mapa a partir de dados topográficos, porém os MGMs apresentam outros fatores como litologia, pedologia, pluviometria e vegetação que também influenciam na estabilidade das encostas (Guidicini e Nieble, 1984).

Como o mapeamento de risco apresenta vários fatores a serem considerados, o que torna as ferramentas de aprendizado de máquina (*Machine Learning* – ML) boas opções de avaliação e predição. Isso ocorre pois conseguem trabalhar com bases de dados com grande dimensionalidade. Diversos trabalhos como Peñafiel e Rojas (2021), Wang et al. (2021), Zydrón et al. (2022) entre outros analisaram a utilização de diferentes tipos de dados em diferentes tipos de ML. Neste trabalho o ML será utilizado para classificar áreas com maior propensão de ocorrer um deslizamento.

1.1. OBJETIVOS

Este trabalho visa, a partir da elaboração de um modelo de aprendizado de máquina, automatizar a produção de mapas iniciais de susceptibilidade à deslizamentos e com essa automatização espera-se identificar de áreas de maior risco para aprofundamento dos estudos e excursões de campo.

1.2. ÁREA DE ESTUDO

1.2.1. Localização

A área de estudo está delimitada pelos municípios de Bertioga e São Sebastião (Figura 1).

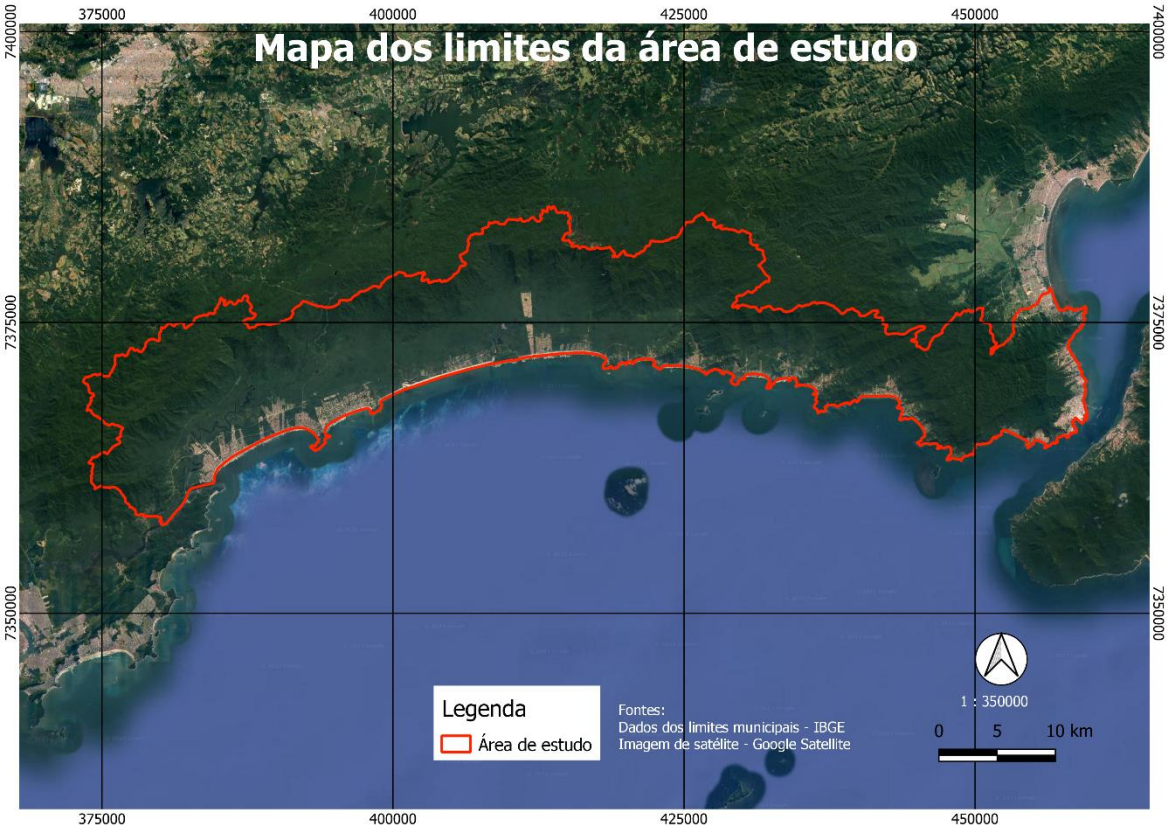

Figura 1: Mapa dos limites da área de estudo

1.2.2. Geomorfologia

Segundo a divisão geomorfológica de Almeida (2018) a área de trabalho está inserida na Província Costeira, contendo a subclassificação de Serrania Costeira - Serra do Mar. Sua topografia varia do nível do mar até 1315 m (Figura 2), as encostas apresentam direcionamento principal em NE-SW (nordeste – sudoeste) (Figura 3) e ângulo de inclinação varia de 0º a 70º (Figura 4).

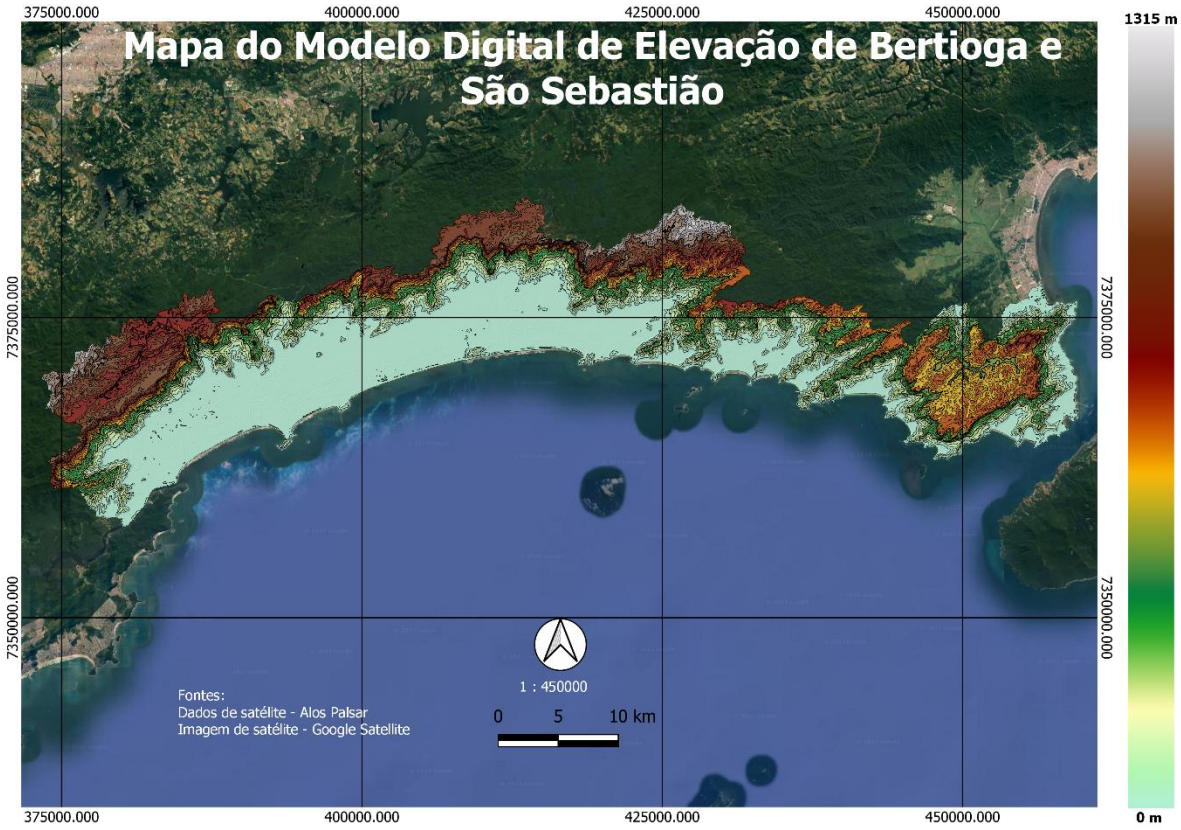

Figura 2: Mapa do Modelo Digital de Elevação

Figura 3: Mapa de Direção das encostas

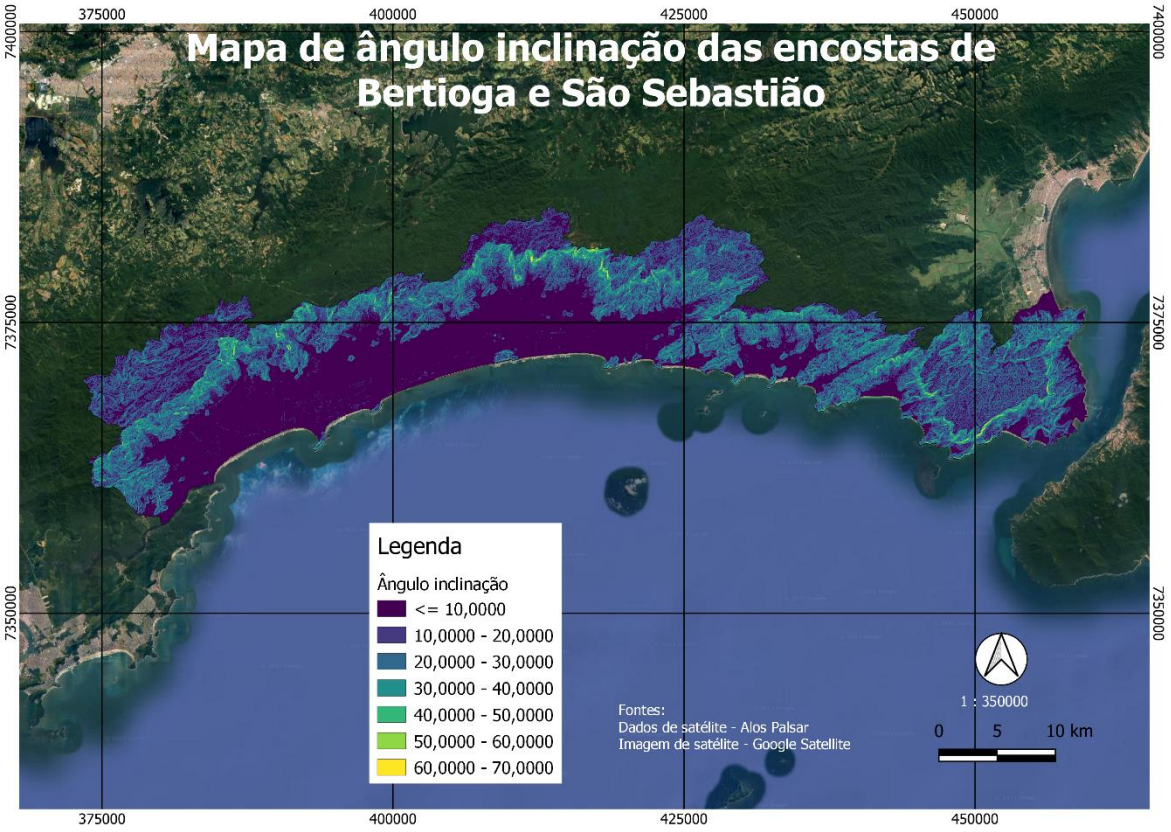

Figura 4

1.2.3. Geologia

A área de trabalho está inserida na Faixa Ribeira, essa que participou do episódio de colagem do cráton São Francisco com o cráton do *Rio de la Plata*, cesse evento originou-se na parte oeste do paleocontinente Gondwana no Neoproterozóico ao Eopaleozóico (Brito Neves e Cordani, 1991).

Localmente, a litologia é dividida em três conjuntos (Almeida, 2018; Dias Neto, 2001) (Figura 5):

1) “Sequências gnáissicos-migmatíticas consistem em ortognaisses e migmatitos com paleossoma de hornblenda-biotita gnaisse e neossoma de composição granodiorítica.”

2) “Rochas paraderivadas constituídas predominantemente por rochas siltitoargilosas e subordinadamente, por quartzitos e rochas calcossilicatadas.”

3) “Rochas granulíticas predominantemente charnockitos e noritos que ocorrem associados aos gnaisses oftalmíticos e aos corpos graníticos presentes na área."

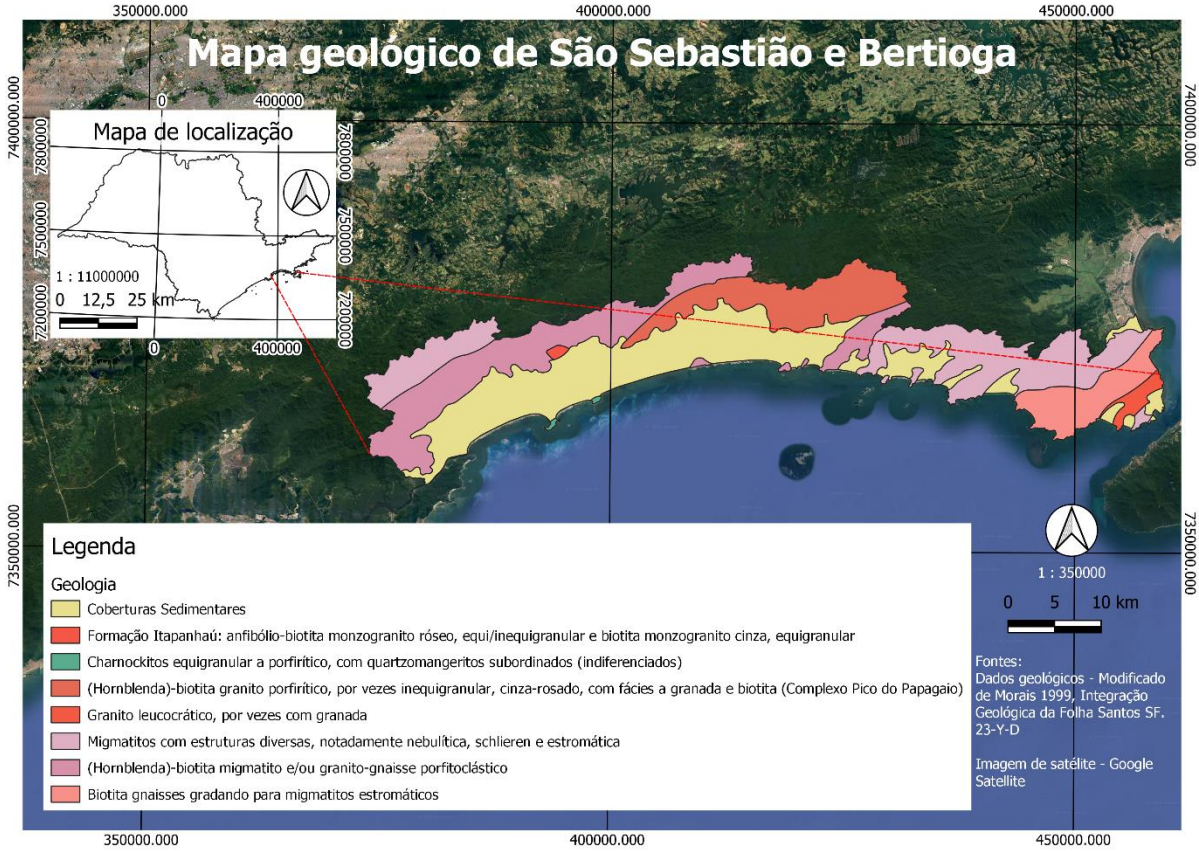

Figura 5: Mapa geológico da área de estudo

1.2.4. Pedologia

Utilizando a classificação de pedologia de Rossi (2017), a área de estudo é caracterizada por Argissolos Amarelos, Argissolos Vermelho-Amarelos, Cambissolos Háplicos, Espodossolos Humilúvicos, Gleissolos Háplicos, Gleissolos Sálicos, Neossolos Flúvicos, Neossolos Litólicos, Neossolos Quartzarênicos e Organossolos Háplicos (Figura 6 e Quadro 1).

Figura 6: Mapa de pedologia da área de estudo

Quadro 1: Tabela dos solos presentes na área de estudo com as descrições de cada tipo (Rossi, 2017).

Unidade de mapeamento	Classes dos solos	Descrição
CX7	CAMBISSOLO HÁPLICO	Associação de CAMBISSOLO HÁPLICO Tb Distrófico A moderado ou proeminente, textura indiscriminada, bem a imperfeitamente drenado + NEOSSOLO FLÚVICO e GLEISSOLO HÁPLICO/MELÂNICO, indiscriminados, todos fase relevo suave ondulado
CX9		Associação de CAMBISSOLO HÁPLICO Distrófico/Eutrófico, típico ou latossólico textura argilosa e média, A moderado e proeminente + ARGISSOLO AMARELO/VERMELHO-AMARELA Distrófica textura média/argilosa e argilosa, não rochoso e rochoso, ambos fase relevo forte ondulado
CX16		Associação de CAMBISSOLO HÁPLICO Tb A moderado ou proeminente + LATOSSOLO AMARELO/VERMELHOAMARELO típico ou cambissólico A moderado, ambos Distróficos, textura média ou argilosa, fase relevo forte ondulado
CX18		CAMBISSOLO HÁPLICO Tb textura média e argilosa, rochoso e não rochoso, fase substrato granitóides, relevo forte ondulado e ondulado
CX20		Associação de CAMBISSOLO HÁPLICO típico, textura argilosa e média, A moderado e proeminente + ARGISSOLO AMARELO/VERMELHO-AMARELO textura média/argilosa e argilosa, não rochoso e rochoso, ambos Distróficos, fase relevo forte ondulado
CX21		Associação de CAMBISSOLO HÁPLICO textura argilosa ou média + NEOSSOLO LITÓLICO textura média, substrato granitoides, ambos Tb Distrófico A moderado, fase relevo forte ondulado e montanhoso
EK	ESPODOSSOLO HUMILÚVICO	ESPODOSSOLO HUMILÚVICO/FERRI-HUMILÚVICO Hidromórfico ou não hidromórfico distrófico, fase relevo plano e suave ondulado
GX2	GLEISSOLO HÁPLICO	Complexo Indiscriminado de GLEISSOLO HÁPLICO ou MELÂNICO com ou sem ocorrência de ORGANOSSOLO, fase relevo plano
GX4		Grupamento indiscriminado de GLEISSOLO HÁPLICO ou MELÂNICO e CAMBISSOLO HÁPLICO Tb Distrófico A moderado ou proeminente, textura indiscriminada, bem a imperfeitamente drenado, todos fase relevo plano
GZ	GLEISSOLO SÁLICO	Associação de GLEISSOLO SÁLICO ou TIOMÓRFICO + NEOSSOLO QUARTZARÊNICO Hidromórfico sálico, fase relevo plano
OX3	ORGANOSSOLO HÁPLICO	Associação de ORGANOSSOLO HÁPLICO + GLEISSOLO HÁPLICO ou MELÂNICO indiscriminados, ambos fase relevo plano
PA	ARGISSOLO AMARELO	Associação de ARGISSOLO AMARELO típico, textura arenosa/média e média/média + NEOSSOLO LITÓLICO típico A moderada textura média e arenosa, substrato arenito, ambos Distróficos, A moderado, fase relevo ondulado
PVA16	ARGISSOLO VERMELHO-AMARELO	ARGISSOLO VERMELHO-AMARELO/AMARELO Distrófico latossólico, A moderada textura argilosa ou argilosa/muito argilosa, fase relevo forte ondulado e ondulado
PVA33		Associação de ARGISSOLO VERMELHO-AMARELO distrófico típico, A moderado ou proeminente, textura média/argilosa, pouco profundo + CAMBISSOLO HÁPLICO A moderada textura argilosa + NEOSSOLO LITÓLICO Eutrófico/Distrófico, textura média substrato sedimentos do Grupo Passa Dois, todos fase relevo ondulado
RL23	NEOSSOLO LITÓLICO	Associação de NEOSSOLO LITÓLICO distrófico típica textura média ou argilosa, com ou em cascalho, fase substrato granitóides, relevo montanhoso e escarpado + Afloramento rochoso
RQ8	NEOSSOLO QUARTZARÊNICO	NEOSSOLO QUARTZARÊNICO Hidromórfico ou Órtico típico, sedimentos marinhos atuais, fase relevo suave ondulado e plano
RY2	NEOSSOLO FLÚVICO	Associação de NEOSSOLO FLÚVICO Psamítico e/ou Tb Distrófico textura média e argilosa, A moderado + GLEISSOLO Indiscriminado substrato sedimentos fluviais, ambos fase relevo plano

1.2.5. Pluviometria

Analisando os hietogramas de Bertioga e São Sebastião (Figura 7 e 8), historicamente, o período entre dezembro e março é o que mais chove. Isso se deve à presença da Serra do Mar que atua como uma barreira forçando o ar úmido a ascender, essa massa de ar resfria de forma adiabática e forma nuvens estratiformes e cumuliformes gerando a precipitação do tipo orográfico (Tavares, 2009).

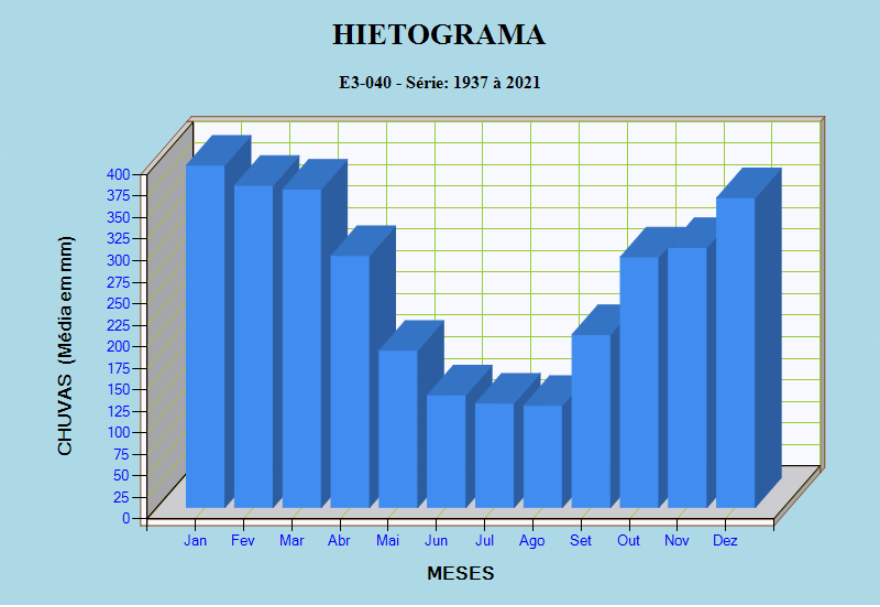

Figura 7: Hietograma do pluviômetro E3-040 (Usina Itatinga; latitude 23° 46' 29"S; longitude 46°06' 38"W; altitude 20 m) no período de 1937 à 2021 (site do Departamento de Águas e Energia Elétrica, acessado 16/11/2023)

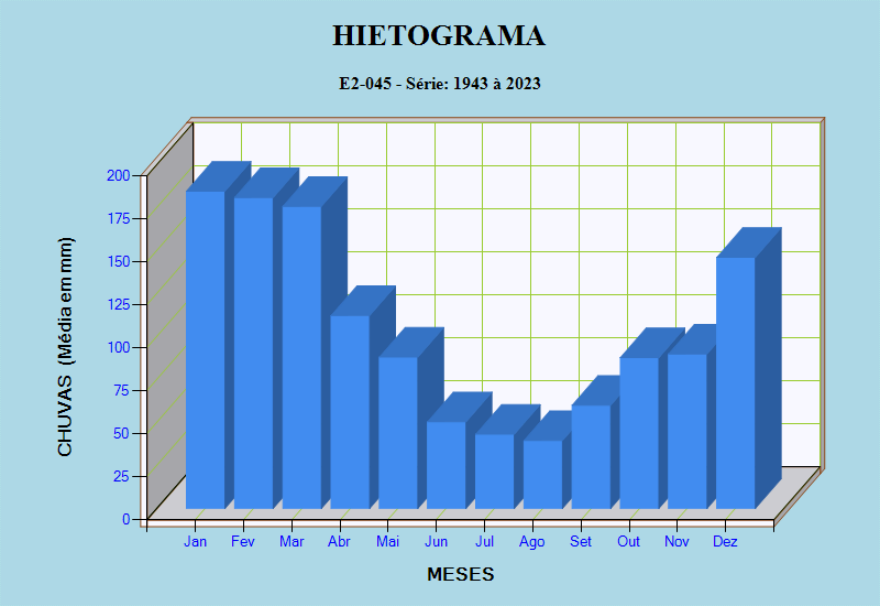

Figura 8: Hietograma do pluviômetro E2-045 (São Francisco; latitude 23° 46' 00"S; longitude 45°25' 00"W; altitude 20 m) no período de 1943 à 2023 (site do Departamento de Águas e Energia Elétrica, acessado 16/11/2023).

1.2.6. Vegetação

Utilizando Brito e Oliveira (2008) em conjunto com o arquivo de polígonos de Nalon et al. (2022), a área de trabalho apresenta vegetações de Floresta Densa Montana, Floresta Densa Sub Montana, Floresta Densa das Terras Baixas, Formação Arbórea/arbustiva-herbácea de Várzea e Formação Arbórea/arbustiva-herbácea de Terrenos Marinhos Lodosos (Figura 9).

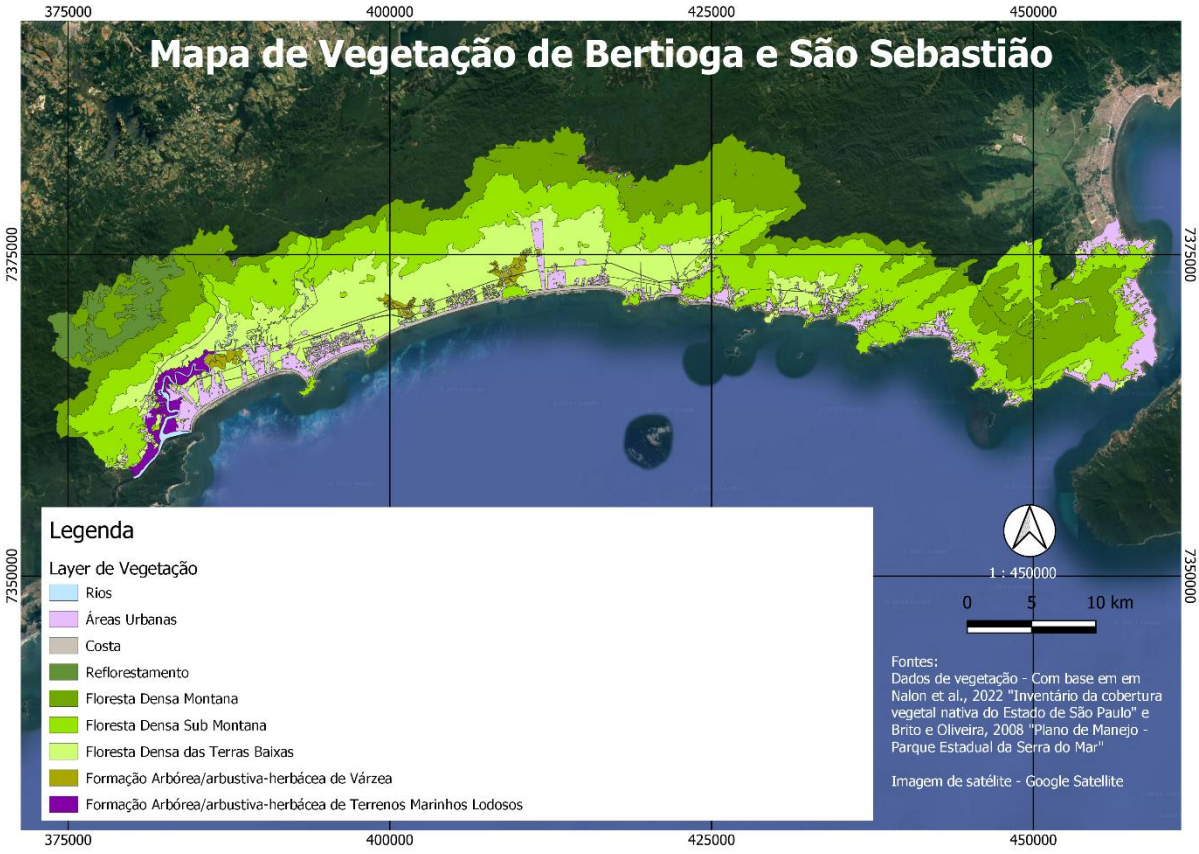

Figura 9: Mapa de vegetação dos municípios de Bertioga e São Sebastião

2. EMBASAMENTO TEÓRICO

2.1. Movimentos gravitacionais de massa e os seus agentes

Os MGMs podem ser caracterizados como “movimento de solo, rocha e/ou vegetação ao longo da vertente, sob a ação direta da gravidade.” (Tominaga et al., 2009, p. 27). Esse material transportado pode atingir infraestruturas e a população criando prejuízos. Por isso, para preveni-

los, trabalhos como os de Terzaghi (1950), Varnes (1978), Guidicini e Nieble (1984), Tavares et al. (2004), Tominaga et al. (2009), Tominaga et al. (2015), Wyllie (2017) e Silva et al. (2022) estudam os mecanismos e os fatores que influenciam nos MGMs.

Neste trabalho foi adotado a classificação dos agentes e causas de Guidicini e Nieble (1984), que apesar de ser antigo ainda se aplica como observado em Tominaga et al. (2009; 2015). Nessa classificação os agentes são separados em predisponentes e efetivos (Quadro 2). O primeiro atua como fator condicionante ou "pano de fundo" enquanto o segundo atua diretamente na desestabilização do talude. Dentre os fatores efetivos ainda há a subclassificação por participação sendo eles os agentes efetivos preparatórios que atuam de forma cumulativa e os agentes efetivos imediatos que atuam instantaneamente.

Quadro 2: Agentes predisponentes e efetivos de eventos de MGM (Guidicini e Nieble, 1984)

Classificação		Elementos
Agentes Predisponentes		• Complexo Geológico, • Complexo Morfológico, • Complexo Climático-Hidrológico, • Gravidade, • Calor Solar, • Tipo de Vegetação Original
Agentes efetivos	**Preparatórios**	• Pluviosidade, • Erosão pela água ou vento, • Congelamento e degelo, • Variação de temperatura, • Dissolução química, • Ação de fontes e mananciais, • Oscilações de nível dos lagos e marés e do lençol freático, • Ação humana e de animais, inclusive desflorestamento.
	Imediatos	• Chuva intensa, • Fusão de gelo e neve, • Erosão, • Terremotos, • Ondas, • Vento, • Ação do homem • Etc

Ainda segundo Guidicini e Nieble (1984), há a classificação de causas internas e externas. As causas internas são aquelas que desestabilizam o talude através da diminuição da resistência interna do material sem alterar a geometria do talude, já as causas externas são

aquelas que alteram a geometria sem influenciar na resistência interna do material. A partir do Quadro 2 e dessa classificação das causas foram selecionados o “Complexo Geológico”, “Complexo Morfológico”, “Tipo de Vegetação Original” e “Pluviosidade”.

O “Complexo Geológico” foi selecionado para representar a importância das características geotécnicas da litologia e da pedologia. Na litologia, características como o tipo de rocha, estruturas (falhas, fraturas, diques etc) e produtos de alteração podem diminuir a resistência do maciço (Wyllie, 2017). Na pedologia, características como tipo de solo, granulometria, porosidade, espessura dos horizontes etc podem diminuir a resistência do maciço (Antunes e Salomão, 2018).

O “Complexo Morfológico”, altera a geometria sem diminuir a resistência interna do material. Por exemplo, quando o ângulo de inclinação do talude supera o ângulo de atrito interno do material, fazendo com que a força cisalhante seja maior que a de atrito interno (Guidicini e Nieble (1984), Wyllie, (2017)). Para o mapeamento de risco do “Manual de mapeamento de perigo e risco a movimentos gravitacionais de massa” de Pimentel e Santos (2018) os dados de topografia são chave para definir as áreas de risco.

A presença da vegetação aumenta a resistência interna do solo, com cada parte da planta influenciando de forma diferente o talude (Guidicini e Nieble , 1984; Stokes et al., 2008; Hairiah et al., 2020):

1) As raízes providenciam ancoragem aumentando a resistência ao cisalhamento e absorvem a água e sais minerais do substrato diminuindo a pressão hidrostática.
2) Os caules retêm o solo erodido.
3) As folhas interceptam a precipitação e iniciam a evapotranspiração diminuindo a umidade do solo.

Outra forma da vegetação contribuir com a estabilização do o talude é com a variedade vegetal (Stokes et al., 2008):

1) Gramíneas: vegetação que cresce e se recupera rápido, apresenta raízes rasas que formam uma rede densa protegendo o solo de erosão superficial, algumas espécies

apresentam raízes mais profundas que são reconhecidas pela restauração de taludes erodido.

2) Herbáceas: vegetação que, assim como as gramíneas, apresenta raízes rasas formando uma rede densa.
3) Arbustivo: vegetação que, dependendo da espécie, apresenta um sistema de raiz com tensão cisalhante comparável com o das árvores.
4) Arbóreo: vegetação que, dependendo do tipo de solo, tem raízes profundas e extensas, o que beneficia no reforço do solo em taludes.

A água influencia na estabilidade do talude através da pressão hidrostática, o que diminui a resistência interna do material sem alterar a geometria. A "pluviosidade" foi selecionada visto que a água demora até sair totalmente do solo, em regiões com pluviosidade alta a água vai se acumulando e desestabilizando o talude (Tavares (2004) e Silva et al. (2022)). Segundo Tavares (2004) a maioria das ocorrências de deslizamentos no litoral norte estão associadas com a chuva acumulada de 120 mm no período de três dias (72 horas). Por isso neste trabalho decidiu-se utilizar um agente efetivo preparatório ao invés do "Complexo Climático-Hidrológico" e da "Chuva Intensa".

2.2. Sensoriamento Remoto

Segundo Meneses e Almeida (2012), "Sensoriamento Remoto é uma ciência que visa o desenvolvimento da obtenção de imagens da superfície terrestre por meio da detecção e medição quantitativa das respostas das interações da radiação eletromagnética com os materiais terrestres.". Tendo em vista a importância da radiação eletromagnética (REM) o estudo do comportamento dual (modelos ondulatório e corpuscular) dela se torna importante.

A partir das formulações de Maxwell (1865), quando uma partícula carregada se movimenta na velocidade da luz são gerados um campo elétrico e um magnético que são perpendiculares à trajetória da partícula e entre si (Figura 10).

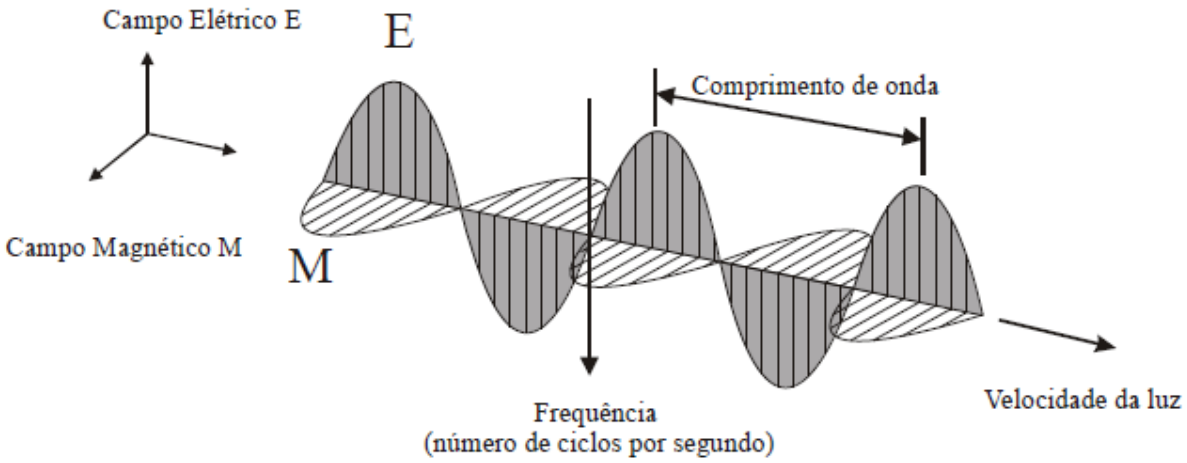

Figura 10: Figura apresentando o modelo ondulatório da REM, retirado de Meneses e Almeida (2012)

Dado o comportamento ondulatório dos campos a equação 1 apresenta a relação entre o comprimento de onda (λ), a velocidade da luz (c) e a frequência (v).

$$\lambda = \frac{c}{v} \qquad (Eq.1)$$

Em 1901, Planck descobriu que a REM transferia quantidades de energia fixa de um corpo a outro, essa energia foi denominada de fóton. A partir dessa descoberta o modelo corpuscular afirma que a energia do fóton, ao interagir com a matéria, é absorvida parcialmente promovendo a mudança da posição do elétron no orbital ou aumentando a intensidade da vibração molecular (Meneses e Almeida, 2012). As equações 2 e 3 apresentam a relação entre a energia (E), constante de Planck (h = 6,624x10-34 Joules*segundos), comprimento de onda (λ), a velocidade da luz (c) e a frequência (v).

$$E = h \times v \qquad (Eq.2)$$

$$E = \frac{h \times c}{v} \qquad (Eq.3)$$

Segundo Meneses e Almeida (2012), a interação entre a REM e o terreno depende do comprimento da onda do raio incidente e do tamanho do objeto analisado, por isso os autores separam a interação entre macroscópica e microscópica. Na interação macroscópica comprimentos de onda maiores que 3 cm resultam em imagens com menos ruídos, uma vez que

evitam a dispersão dos raios em objetos menores (Meneses e Almeida, 2012). As interações microscópicas seguem o modelo corpuscular, em que as REMs com comprimentos de ondas diferentes têm sua energia absorvida de forma diferente o que ressalta o contraste entre os materiais.

As REMs podem vir de fontes naturais com o Sol e a Terra, mas também podem ser artificiais. No sensoriamento remoto a principal fonte de REM natural é o Sol uma vez que sua temperatura, na superfície, é próxima a 6000ºC. A partir da lei de Planck (equação 4) foi obtida a Figura 11 que apresenta a distribuição de energia de corpos negros a diferentes temperaturas em diferentes comprimentos de onda (Meneses e Almeida, 2012).

$$E_\lambda = \frac{2\pi hc^2}{\lambda^5 \left[exp^{\left(\frac{ch}{\lambda KT}\right)} - 1 \right]} \qquad (Eq.4)$$

Onde, $E_\lambda = energia\ radiante\ espectral\ medida\ em\ W\ m^{-2}\ \mu m^{-1}$

$h = constante\ de\ Planck = 6{,}6256\ \times 10^{-34}\ W\ s^2$

$c = velocidade\ da\ luz = 2{,}997925 \times\ 10^8\ m\ s^{-1}$

$K = constante\ de\ Boltzman = 1{,}38054 \times 10^{-23}\ W\ s\ K^{-1}$

$T = temperatura\ absoluta\ da\ fonte\ K$

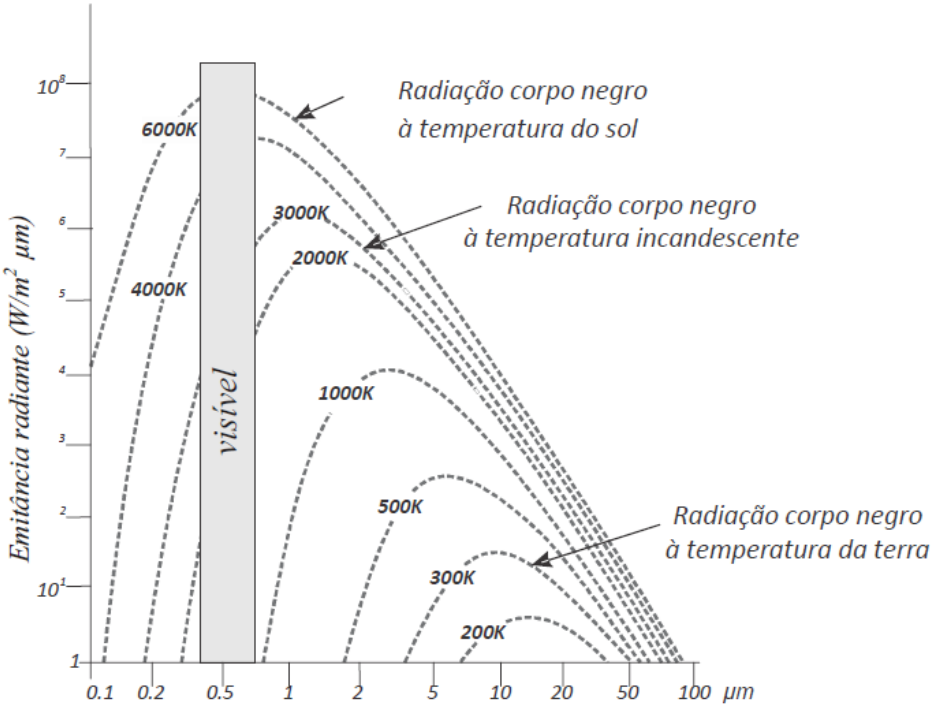

Figura 11: Comportamento espectral da Lei de deslocamento de Wien (Meneses e Almeida, 2012).

As REMs de fontes naturais conseguem atingir no máximo comprimentos de onda de 100 micrômetros (Figura 11), por isso para obter REMs de comprimentos de micro-ondas são utilizadas fontes artificiais. Tendo isso em vista os produtos que utilizam o Sol como fonte trabalham nas faixas espectrais no visível (0,45 - 0,76 μm), Infravermelho próximo (0,76 - 1,2 μm) e Infravermelho de ondas curtas (1,2 – 3,0 μm); enquanto que o espectro das micro-ondas (3,0 – 100 cm) é registrado pelos sensores de Detecção de Ondas de Rádio e Posição (*Radio Detection And Ranging* – RADAR) (Meneses e Almeida, 2012). Para esses dois tipos de sensoriamento são utilizados sensores diferentes, respectivamente, os sensores multiespectrais e os radares de abertura sintética (*Synthetic Aperture Radar* - SAR).

Com o sensoriamento multiespectral são realizadas análises composicionais, uma vez que o comprimento de onda é menor. Das possíveis análises este trabalho utilizou-se a composição de falsa cor e o índice de vegetação de diferencial normalizada (*Normalized Difference Vegetation Index* – NDVI). A composição de falsa cor facilita visualizar o contraste entre materiais diferentes já o NDVI criado por Rouse Junior et al. (1973) utiliza a equação 5 para calcular um valor que representa a saúde das plantas (Figura 12).

$$NDVI = \frac{\text{banda do infravermelho próximo} - \text{banda vermelho}}{\text{banda do infravermelho próximo} + \text{banda vermelho}} \qquad (Eq. 5)$$

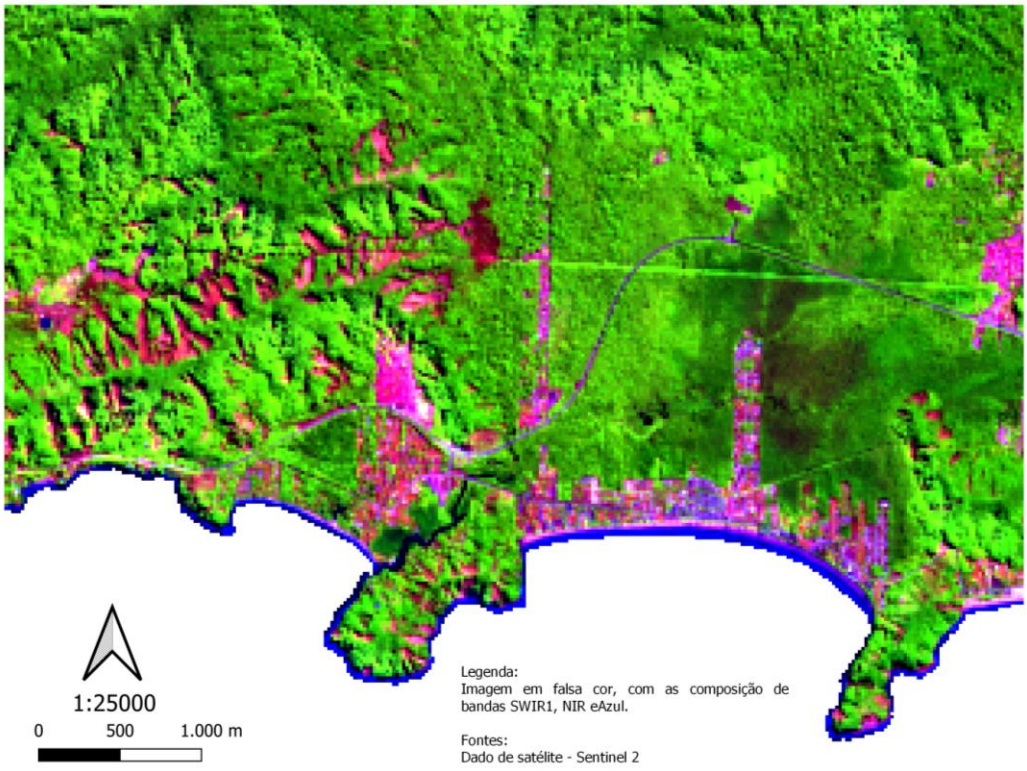

Figura 12: Imagem apresentando o contraste entre a vegetação (verde), solo (rosa escuro) e a urbanização (rosa claro), a composição em cor falsa (SWIR1, NIR e Azul).

Apesar do SAR conseguir produzir imagens da rugosidade do terreno, é necessário a utilização da Interferometria de SAR (InSAR) para coletar os valores de elevação. O InSAR utiliza dois sensores SAR a uma distância (Δx) para criar o efeito de paralaxe o que faz com que os sinais recebidos tenham fases diferentes ($\Delta\varphi$) podendo, assim, calcular a elevação (h) do ponto (Figura 13 e equações 6 a 8) (Meneses e Almeida, 2012).

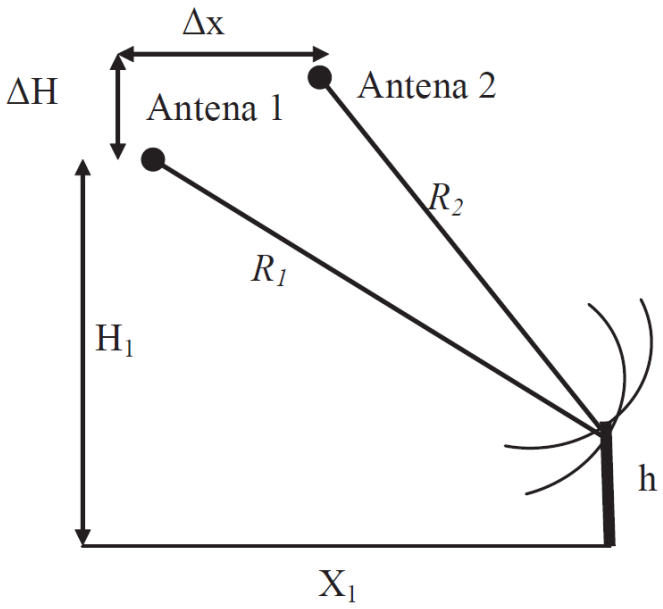

Figura 13: Modelo de imageamento por InSAR (Meneses e Almeida, 2012).

$$\Delta\varphi = -\frac{4\pi}{\lambda}(R_2 - R_1) \qquad (Eq.6)$$

$$R_1 = \sqrt{{x_1}^2 + (H_1 - h)^2} \qquad (Eq.7)$$

$$R_2 = \sqrt{(x_1 - \Delta x)^2 + (H_1 + \Delta H - h)^2} \qquad (Eq.8)$$

Onde, $\lambda = comprimento\ de\ onda\ da\ REM\ incidente$

$R_1 = distância\ entre\ a\ antena\ 1\ e\ o\ ponto$

$R_2 = distância\ entre\ a\ antena\ 2\ e\ o\ ponto$

$H_1 = altitude\ da\ antena\ 1$

$h = altitude\ do\ ponto$

$\Delta H = diferença\ de\ altitude\ da\ antena\ 1\ para\ 2$

$x_1 = distância\ horizontal\ entre\ a\ antena\ 1\ e\ o\ ponto$

$\Delta x = distância\ horizontal\ entre\ a\ antena\ 1\ e\ 2$

2.3. Aprendizado de máquina e aprendizado profundo

Segundo Goodfellow et al. (2016), o aprendizado de máquina (ML – *Machine Learning*) é a capacidade da inteligência artificial (IA) adquirir conhecimento extraindo padrões de dados brutos. Porém para conceitos complexos, como fotos, o aprendizado profundo (DL – *Deep Learning*) é utilizado por extrair padrões a partir de representações mais simples (Goodfellow

et al., 2016). A Figura 14 exemplifica esse conceito, mostrando que a cada camada oculta é extraída um padrão mais complexo.

Figura 14: Imagem representando o aumento da complexidade do padrão que cada camada oculta consegue consegue extrair extraído de Goodfellow et al. (2016).

Neste trabalho foi utilizado o modelo de *perceptron* multicamadas (MLP – *Multilayer Perceptron*), um modelo de aprendizado profundo que utiliza o *perceptron* como um neurônio do modelo. O *perceptron* criado por Rosenblatt (1958) utiliza o neurônio da biologia como base, funcionando da seguinte forma (Figura 15):

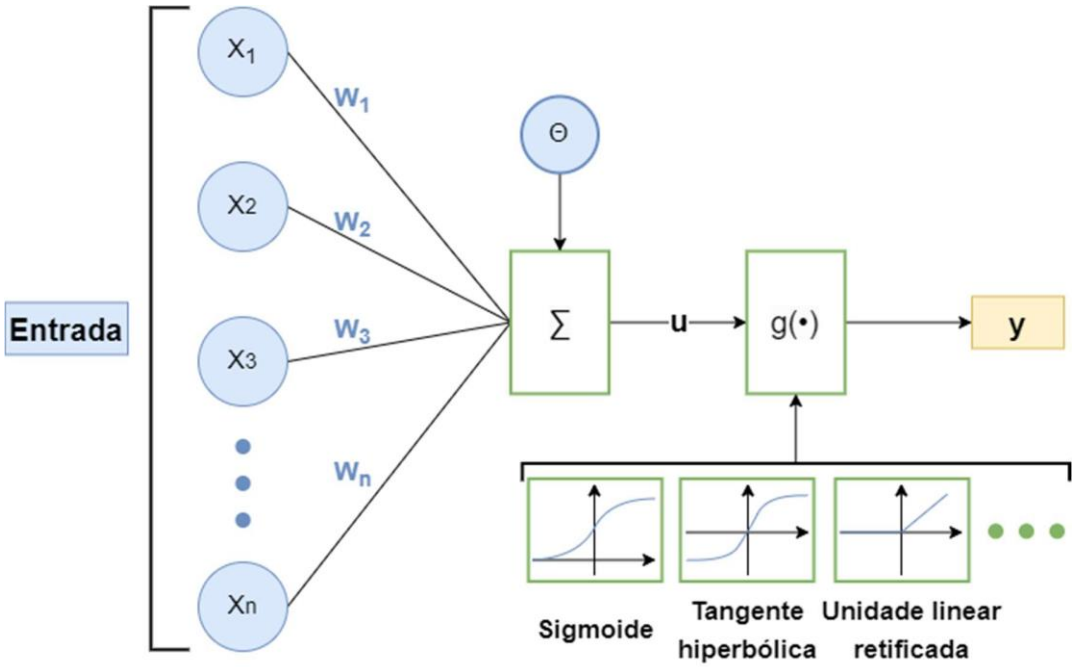

Figura 15 – Figura representando o funcionamento de um Perceptron (extraído de Peñafiel e Rojas (2021))

1) Assim como apresentado na Figura 15, cada *perceptron* recebe informações de uma ou mais entradas e realiza uma soma ponderada com o valor de entrada (x_i), um peso (w_i) e um valor de limiar (θ) (equação 9).

$$u = \sum_i^n (x_i \times w_i) + \theta \qquad (Eq. 9)$$

2) O resultado *u* é então inserido em uma função de ativação *g*(•), essa pode ser do tipo degrau ou degrau bipolar limitando o resultado entre 0 a 1 e -1 a 1, respectivamente. A equação 10 representa a fórmula final para um *perceptron* com resultado *y*.

$$y = g\left(\sum_i^n (x_i \times w_i) + \theta\right) \qquad (Eq. 10)$$

Ao juntar vários *perceptrons*, em paralelo, é formada uma camada; ao empilhar várias camadas é criado uma rede neural artificial que no caso é um MLP, a Figura 16 apresenta a estrutura básica de um MLP.

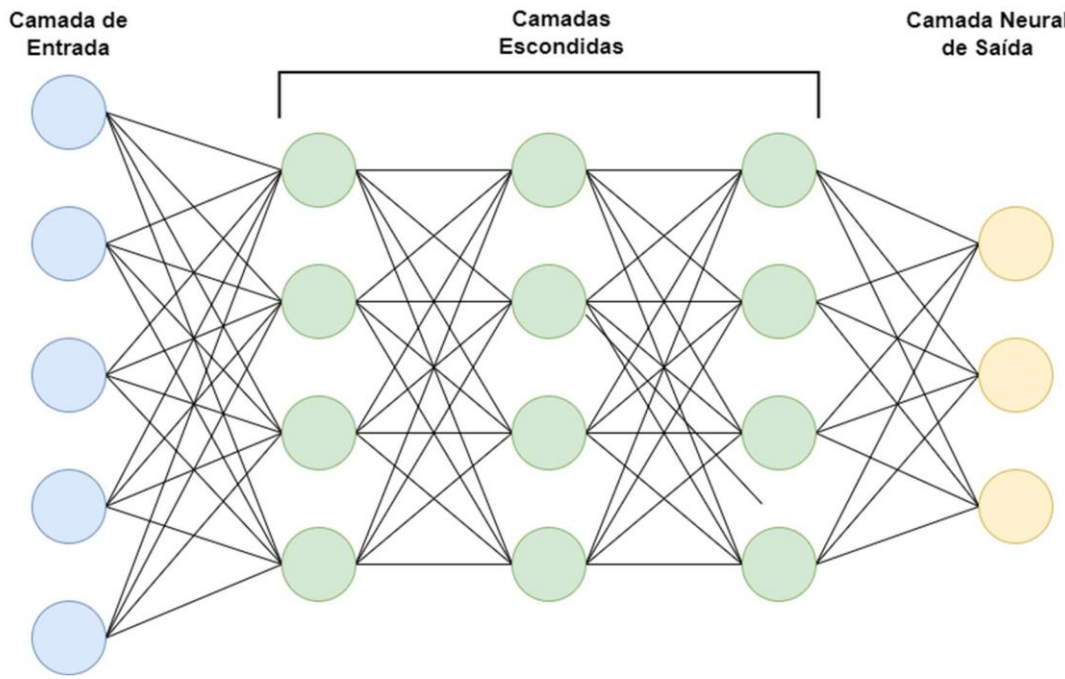

Figura 16: Arquitetura clássica de um MLP (extraído de Peñafiel e Rojas (2021))

Os MLPs são considerados redes neurais artificiais profundas *feedforward* (Goodfellow et al., 2016), ou seja, são redes neurais artificiais em que a informação propaga no sentido da camada de entrada para a de saída. Como o MLP é um modelo com aprendizado supervisionado, quando a informação propaga até a camada de saída é realizado o cálculo do erro a partir de uma função de custo. Segundo Goodfellow et al. (2016), a função custo para as redes neurais, nas maiorias dos casos, utiliza a entropia cruzada dos dados de treinamento e as previsões do modelo.

Para o aprendizado de modelos de MLPs é utilizado o algoritmo de retropropagação (Rumelhart et al., 1986). Esse algoritmo utiliza o valor de erro e propaga no sentido contrário ajustando os valores de peso (w_i) diminuindo, assim, o valor do erro.

3. MATERIAIS E MÉTODOS

3.1. Ferramentas

As ferramentas utilizadas neste trabalho podem ser divididas nas seguintes etapas. Na primeira etapa foram utilizados o Google Colab (Colab) com o a biblioteca do geemap (Wu, 2020) para a aquisição das imagens de sensoriamento remoto, enquanto o QGIS 3.28.12 foi utilizado para trabalhar com os dados de imagens e vetores. Na segunda etapa foi utilizado o

Colab com a biblioteca scikit-learn (Pedregosa et al., 2011) para o treinamento e teste do modelo de ML.

No Colab foram utilizadas outras bibliotecas além das apresentadas, a tabela 1 apresenta elas e suas funções dentro do trabalho.

Tabela 1: Tabela de todas bibliotecas utilizadas e suas funções utilizadas

Biblioteca	Função
OS	Conectar o notebook com o sistema operacional (neste trabalho o google drive)
Numpy	Trabalhar com funções matemáticas e matrizes
Pandas	Trabalhar com tabelas
Matplotlib	Plotar imagens e gráficos
Geopandas	Trabalhar com dados geoespaciais que estão em tabela
Rasterio	Trabalhar com rasters
Scikit Learn	Trabalhar com ML
Imblearn	Trabalhar com bases de dados desbalanceados
ee	Trabalhar com a base de dado e as funções do GEE
IPython	Plotar tabelas
datetime	Trabalhar com datas
geemap	Trabalha com a biblioteca ee e apresenta um mapa interativo

3.2. Dados Base

Como apresentado anteriormente os agentes causadores de MGMs escolhidos foram: “Complexo Geológico”, “Complexo Morfológico”, “Tipo de Vegetação Original” e “Pluviosidade”. A Figura 17 apresenta um fluxograma que associa cada dado base com seu agente correspondente.

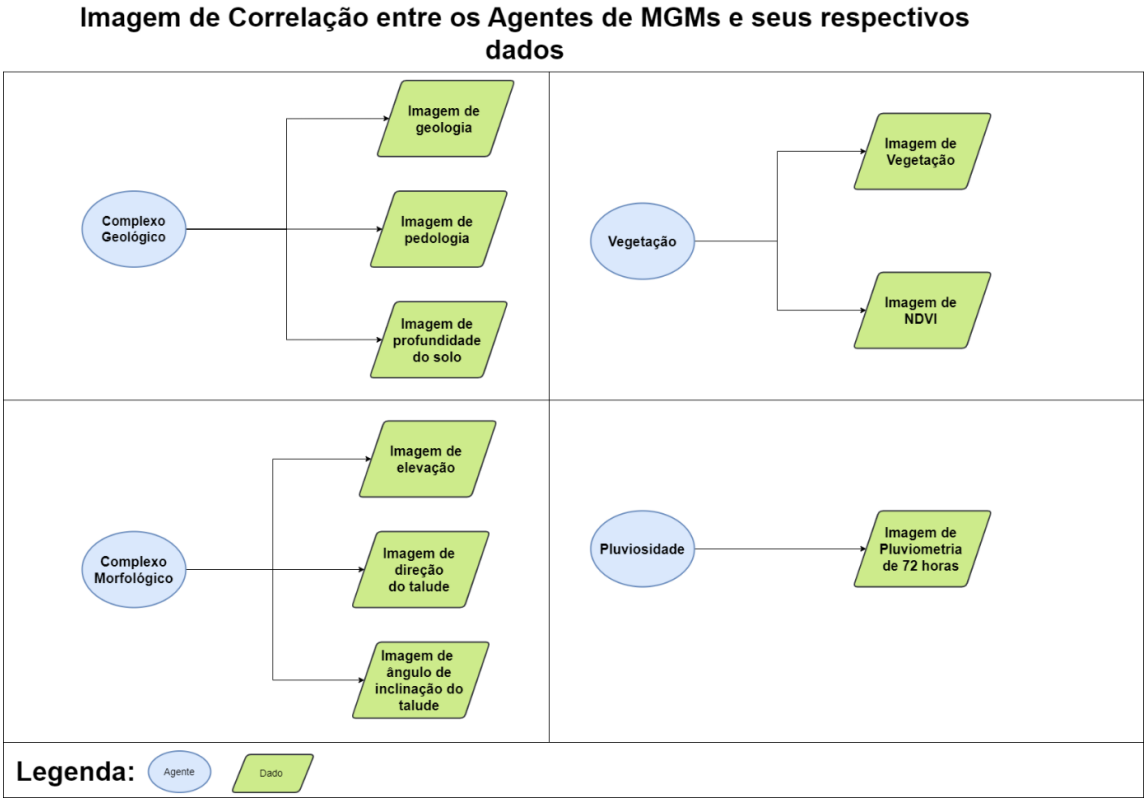

Figura 17: Fluxograma da correlação entre os agentes escolhidos e os dados correspondentes

Dentre os dados apresentados na Figura 17 a "Máscara Binária" não representa nenhum agente, apesar disso ela é necessária para o cálculo da função custo uma vez que representa as ocorrências de MGMs tornando-se assim o arquivo *label*, arquivo esse que serve para avaliar as previsões do modelo.

Como a etapa seguinte utiliza imagens, todos os dados tiveram que ser retrabalhados, tendo em vista que cada dado foi produzido de forma diferente. Por isso as etapas para transformá-los foram diferentes, abaixo estão listados cada dado com as etapas utilizadas:

1) Imagem de geologia (Figura 1): baseado em no mapa de Morais (1999), foi utilizado o QGIS para georreferenciar o mapa e produzir o arquivo de polígonos, finalmente nesse arquivo foi utilizado a ferramenta de rasterização do QGIS para transformar em uma imagem com a resolução de pixels de 125 x 125m.

2) Imagem de pedologia (Figura 2) e Imagem de profundidade de solo (Figura 18): foi utilizado o arquivo de polígonos de pedologia de Rossi (2017), para a produção da imagem foi utilizada a ferramenta de rasterização do QGIS para transformar em uma imagem com a resolução de pixels de 125 x 125m.

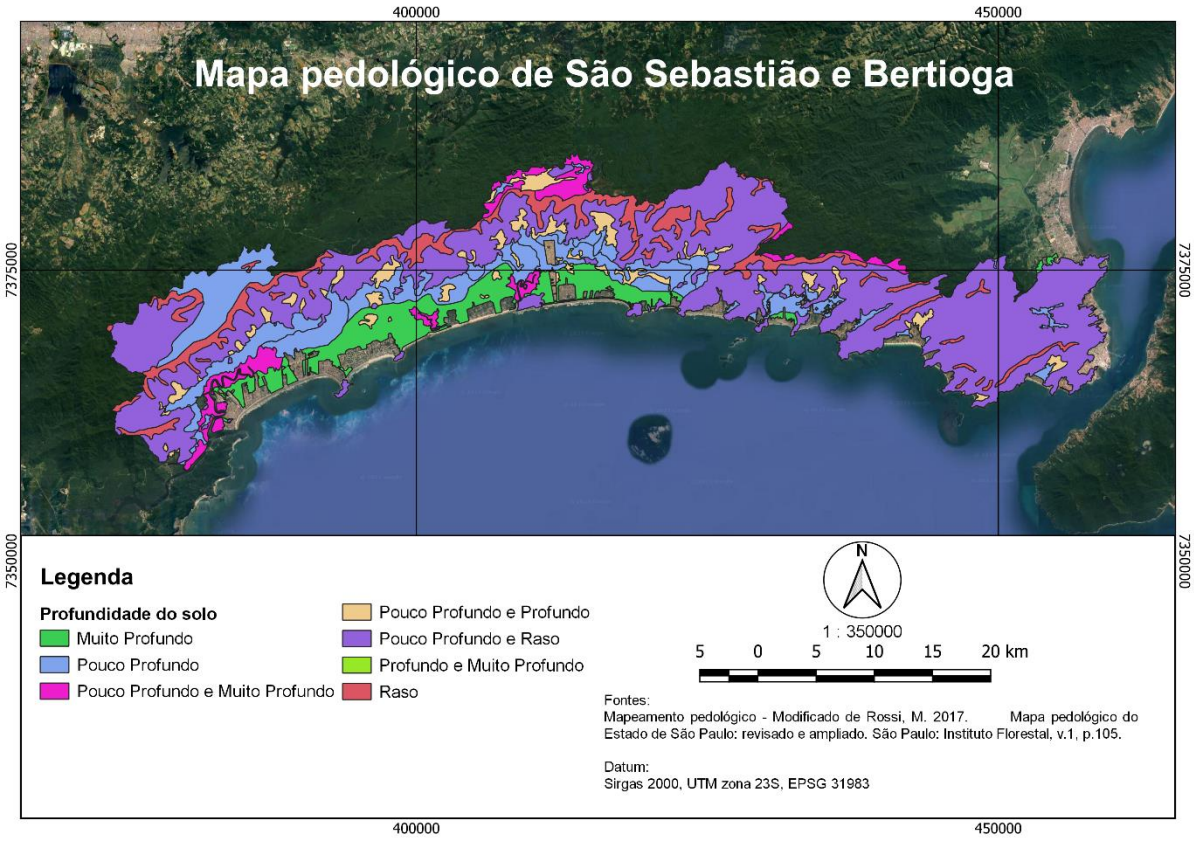

Figura 18: Mapa de profundidade do solo

3) Imagem de elevação (Figura 3): utilizando o código do anexo A obteve-se a imagem de elevação anual do satélite Alos Palsar (2011) com resolução de pixels de 30 x 30m.

4) Imagem de direção de inclinação (Figura 4) e Imagem de ângulo de inclinação (Figura 5): a partir da imagem de elevação obtido anteriormente foram utilizadas as ferramentas de *aspect* e *slope*, respectivamente, para a produção das imagens (resolução de pixels 30 x 30m).

5) Imagem de pluviometria (Figura 19): Definiu-se as datas de aquisição dos dados pluviométricos (16/02/2023, 17/02/2023 e 18/02/2023), utilizando-as como filtro encontrou-se os pluviômetros no site do DAEE (Tabela 2). Os pluviômetros escolhidos se encontram tanto dentro quanto fora da área de trabalho (Figura 20), pois os do lado de fora servem para fechar a imagem durante a interpolação (ferramenta utilizada *IDW interpolation* do programa QGIS). A resolução de pixels da imagem foi definida como a de 30 x 30 m.

Figura 19: Mapa de distribuição dos pluviômetros

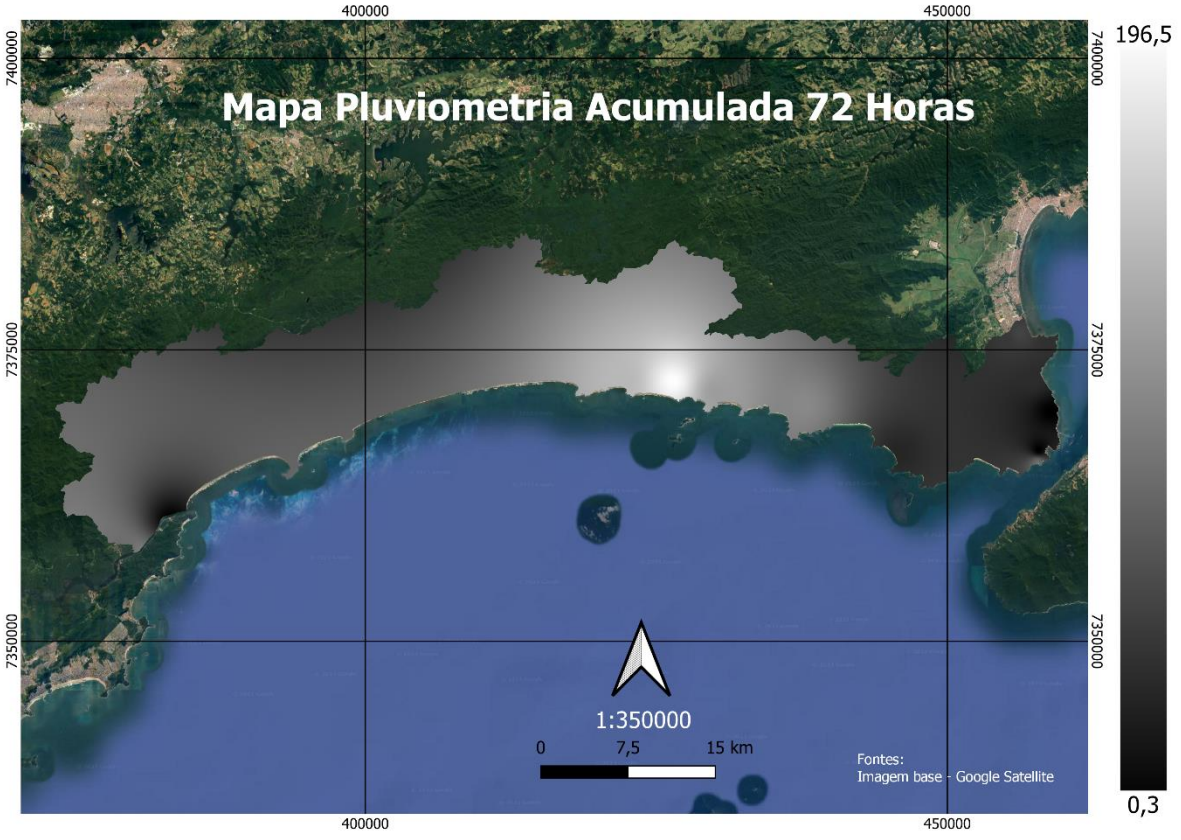

Figura 20: Mapa de Pluviometria acumulada de 72 horas

Tabela 2: Tabela de pluviômetros e pluviometria

Município	Prefixo	Nomes dos pluviômetros	X	Y	Proprietário	Pluviômetro (18/02)	Pluviômetro (17/02)	Pluviômetro (16/02)	Acumulado (72h)
São Sebastião	354850009A	Barra do una	-45.764	-23.758	CEMADEN	127.427	2.413	12.158	141.998
São Sebastião	355070405A	pontal da cruz	-45.404	-23.783	CEMADEN	0.034	0.068	0.088	0.19
São Sebastião	355070409A	itatinga	-45.414	-23.813	CEMADEN	0	0	-	0
São Sebastião	355070411A	juquehy2	-45.722	-23.759	CEMADEN	174.093	2.582	19.948	196.623
São Sebastião	355070414A	boicucanga	-45.612	-23.777	CEMADEN	84.411	3.748	17.784	105.943
São Sebastião	355070420A	praia das cigarras	-45.403	-23.732	CEMADEN	23.128	14.784	0.054	37.966
São Sebastião	355070403A	jaraguá	-45.441	-23.729	CEMADEN	24.901	9.309	9.503	43.713
São Sebastião	355070407A	morro do abrigo	-45.418	-23.76	CEMADEN	22.488	11.045	1.182	34.715
São Sebastião	355070410A	juquehy	-45.685	-23.768	CEMADEN	95.451	2.764	22.601	120.816
São Sebastião	355070413A	toque toque pequeno	-45.531	-23.814	CEMADEN	4.611	1.793	23.324	29.728
São Sebastião	355070418A	praia grande	-45.415	-23.823	CEMADEN	42.034	5.539	39.081	86.654
Salesópolis	354500101A	Ponte Nova	-45.972	-23.575	CEMADEN	0	0	0	0
Caraguatatuba	351050003A	Rio Claro	-45.468	-23.708	CEMADEN	1.653	4.958	11.916	18.527
Caraguatatuba	355070404A	Enseada	-45.43	-23.723	CEMADEN	39.157	9.081	5.121	53.359
Caraguatatuba	351050015A	Fazenda Serra Mar	-45.518	-23.661	CEMADEN	0	0	0	0
Biritiba Mirim	1001286	Barragem Biritiba Montante	-46.088	-23.629	SAISP	0.118	0.016	0.056	0.19
Biritiba Mirim	498	RADAR	-45.971	-23.6	SAISP	0.214	0.002	0.106	0.322
Mogi das Cruzes	350660701A	Jardim Jungers	-46.044	-23.587	CEMADEN	0.074	0.004	0.03	0.108
Santos	354850009A	Carvara	-46.189	-23.891	CEMADEN	56.335	0.042	21.756	78.133
Guarujá	351870107A	morrinhos	-46.26	-23.968	CEMADEN	201.825	0.002	10.467	212.294
Guarujá	351870112A	pereque3	-46.146	-23.864	CEMADEN	0.006	0.006	0.018	0.03
Guarujá	351870115A	pereque	-46.186	-23.938	CEMADEN	138.119	4.925	15.029	158.073
Guarujá	354850014A	sitio das neves	-46.314	-23.881	CEMADEN	0	0	0	0
Guarujá	351870108A	jardim da esperança	-46.284	-23.959	CEMADEN	166.278	0.002	10.431	176.711
Guarujá	351870114A	balnerário pernambuco	-46.191	-23.97	CEMADEN	184.074	0.002	33.548	217.624
Guarujá	351870116A	pereque 2	-46.164	-23.907	CEMADEN	129.312	0.789	21.757	151.858

Fonte: Departamento de Águas e Energia Elétrica

6) Imagem de vegetação (Figura 9): Foram utilizados os polígonos do Nalon et al. (2022) como base para seguintes limites:
    - Infraestruturas,
    - Rios,
    - Praia e mar,
    - Floresta Densa das Terras Baixas.

    Já o trabalho do Brito e Oliveira (2008) definiu os seguintes limites:
    - Floresta Densa Montana
    - Floresta Densa Sub Montana
    - Formação Arbórea/arbustiva-herbácea de Várzea
    - Formação Arbórea/arbustiva-herbácea de Terrenos Marinhos Lodosos

    Por fim, foi utilizada a ferramenta de rasterização do QGIS resolução de pixels de 30 x 30m.

7) Imagem de ndvi (Figura 21): Utilizando o código do anexo A, foi obtido a imagem do satélite Sentinel 2 (10/02/2023) e com a biblioteca geemap realizou-se o cálculo de bandas, a imagem final apresenta resolução de pixels de 20 x 20m.

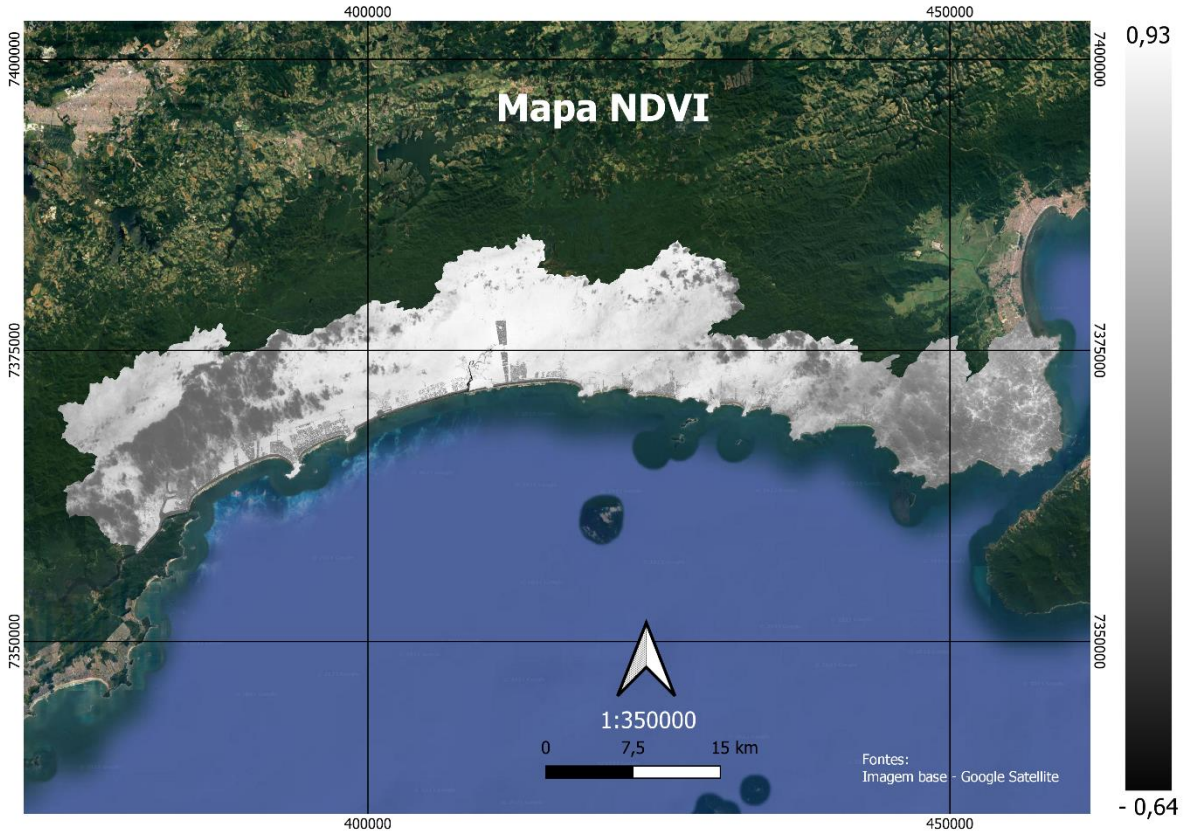

Figura 21: Mapa de NDVI

8) Imagem da máscara binária (Figura 22): Utilizando o código do anexo A, foi a obtida a imagem do satélite Sentinel 2 (01/04/2023) e com a biblioteca geemap realizou-se a composição de cor falsa (infravermelho de ondas curtas 1, infravermelho próximo e azul), com essa imagem realizou-se a delimitação das ocorrências. Por fim, foi utilizada a ferramenta de rasterização com a imagem final apresentando a resolução dos pixels de 20 x 20m. A máscara binária está apresentada com dimensões reduzidas, pois só foram registradas as ocorrências nessa área, por conseguinte a imagem da máscara binária foi a única a ser tratada assim.

Figura 22: Mapa da máscara binária, a área escura representam o local onde não houveram deslizamentos, enquanto a área branca são as ocorrências.

3.3. Machine Learning

O Anexo B apresenta o código utilizado para realizar a etapa de ML. Esta etapa foi separada em duas partes, a primeira de pré-processamento e a segunda de treinamento do modelo.

Devido à concentração de pontos de ocorrência na área central a área de trabalho foi dividida em três partes iguais para evitar enviesamento, uma vez que as ocorrências registradas estão concentradas na área central. Por isso os dados de entrada utilizados para treinar o modelo foram os da área central Figura 23.

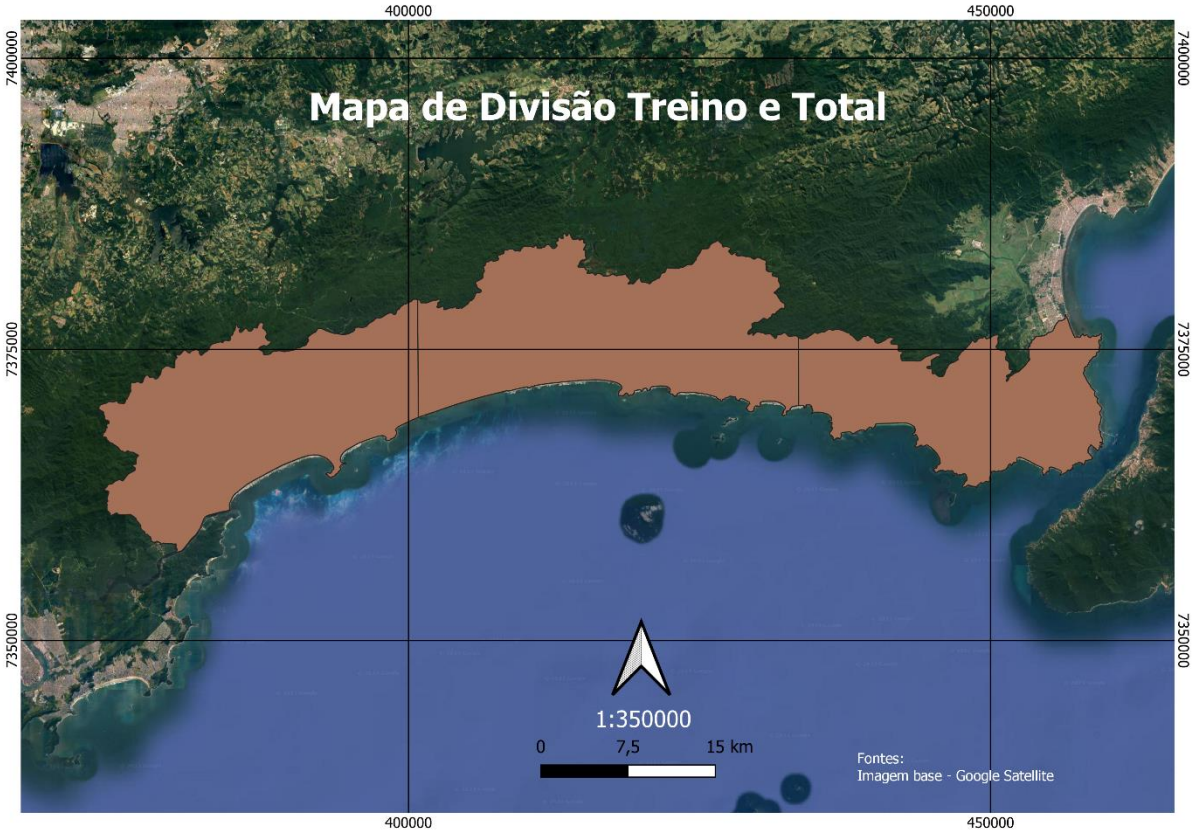

Figura 23: Mapa de separação da área de trabalho em treino e total

Para a etapa de pré-processamento foi necessário conferir alguns metadados das imagens conforme a Tabela 3. Observando a tabela 3 cada imagem apresenta dimensões e valores mínimos e máximo dos pixels diferentes. Por isso foram montadas as funções redimensionarImagem(), normalizarDados() e preparandoDados() que, respectivamente, redimensionam, normalizam e transformam os dados no formato utilizado para treinar o modelo.

Tabela 3: Metadados das imagens com as dimensões e os valores

Dados	Largura	Altura	Proporção	Fator largura	Fator altura	Valor min	Valor max
Máscara Binária	1637	791	2.07	0.16	0.16	0.00	1.00
Geologia	262	126	2.08	1.00	1.00	1.00	35.00
Pedologia profundidade	262	126	2.08	1.00	1.00	0.00	7.00
Pedologia	262	126	2.08	1.00	1.00	0.00	17.00
Vegetação	1090	526	2.07	0.24	0.24	1.00	8.00
Topografia	1092	528	2.07	0.24	0.24	0.00	1313.31
Direção de inclinação	1092	528	2.07	0.24	0.24	0.00	57.15
Ângulo de inclinação	1092	528	2.07	0.24	0.24	1.94	358.41
Pluviometria Acumulada 72H	1092	528	2.07	0.24	0.24	39.68	196.55
NDVI	1638	792	2.07	0.16	0.16	0.00	0.93

Antes de realizar o redimensionamento dos dados foram calculados os fatores de redimensionamento (Tabela 3, colunas “Fator largura” e “Fator altura”). Com esses valores a função redimensionarImagem() utiliza a ferramenta *Resampling* da biblioteca Rasterio para interpolar e redimensionar as imagens. Porém, as imagens de máscara binária, geologia, pedologia profundidade, pedologia e vegetação são categóricas, enquanto as de topografia, direção de inclinação, ângulo de inclinação, pluviometria acumulada 72H e NDVI são contínuas. Por isso, como a figura 24 apresenta, para os dados categóricos foi utilizado a interpolação por vizinho mais próximos, já que mantêm os valores originais; enquanto que para os dados contínuos foi utilizada a interpolação bilinear que evita artifícios que a interpolação por vizinhos mais próximos gera (Gonzalez e Woods, 2018)

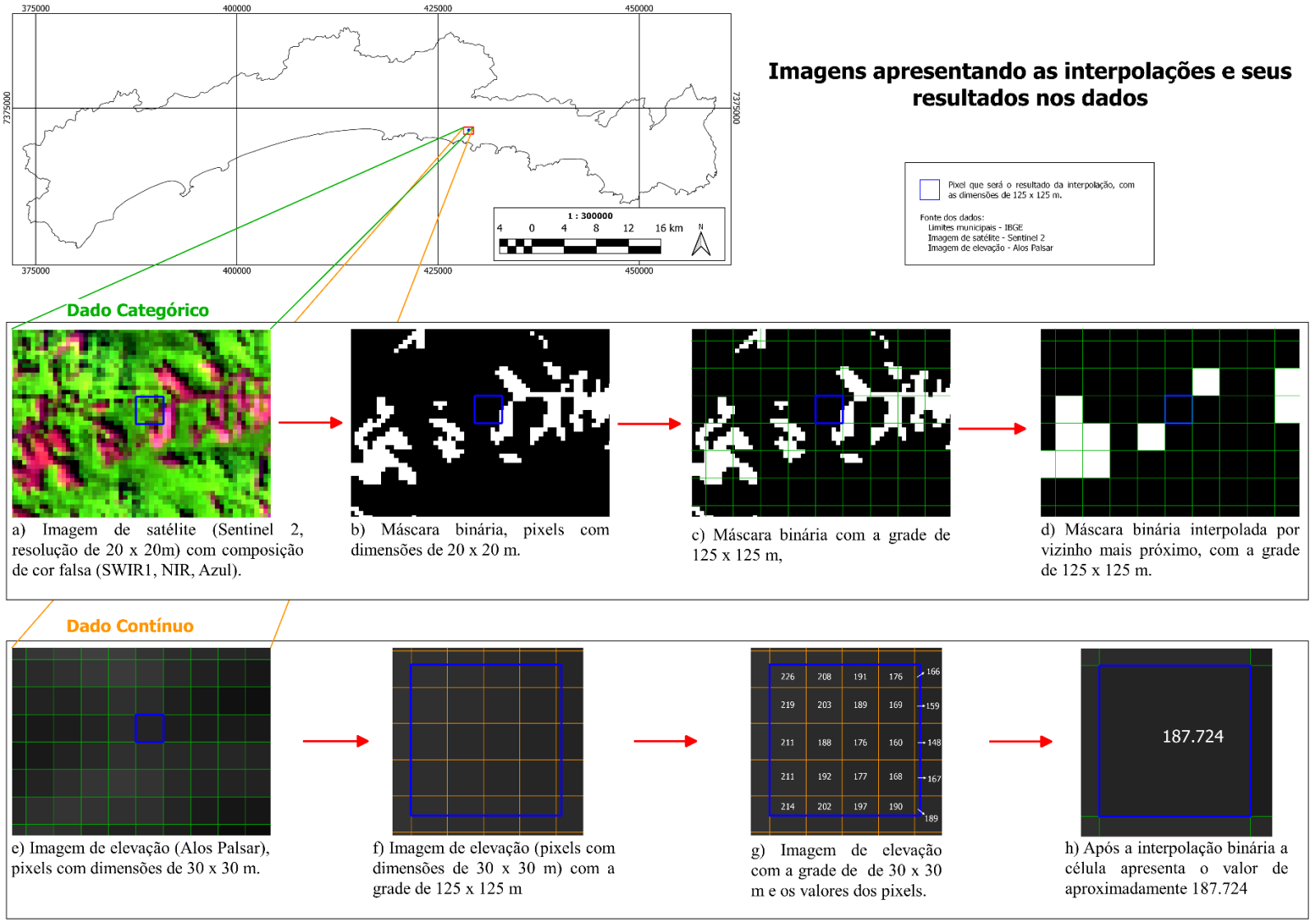

Figura 24:Imagem apresentando as diferentes interpolações utilizadas com os dados do trabalho

Antes de normalizar os valores dos pixels os dados que estavam no formato de imagem foram convertidas em *arrays* e depois em um *ndarray*. O *array* é um formato equivalente a uma matriz o que facilita realizar operações matemáticas, enquanto o *ndarray* é o empilhamento das *array* formando uma *array* tridimensional. Neste trabalho o *ndarray* apresenta 10 *arrays* (dimensões) com cada uma apresentando 126 linha e 262 colunas.

A função normalizarDados(), utiliza a equação 12 como base para normalizar as imagens.

$$Valor\ normalizado = \frac{valor\ do\ pixel - valor\ mínimo}{valor\ máximo - valor\ mínimo} \qquad Eq.12$$

Com os dados normalizados a função preparandoDados() realiza o enfileiramento dos valores que , como o nome indica, enfileira todas da linhas de cada *array* (Figura 25) fazendo com que cada uma tenha dimensões de 1 linha por 33012 colunas.

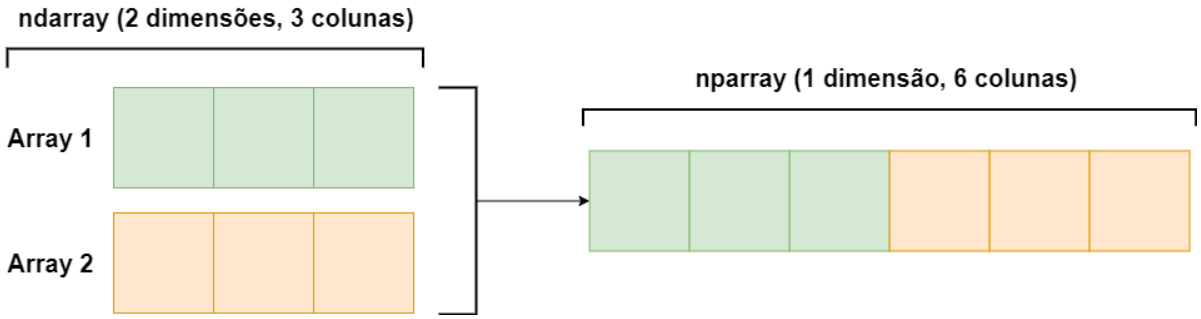

Figura 25: Imagem que exemplifica o enfileiramento de um ndarray com (2 dimensões e 3 células) para um de uma dimensão por 6 colunas.

Após a etapa de pré-processamento foram utilizados as funções separandoTreinoTeste() e definindoHiperparametros(). A função separandoTreinoTeste() utiliza as ferramentas *RandomUnderSampler* e *train_test_split*. O *RandomUnderSampler* foi utilizado para "balancear" os dados, pois retira valores da classe majoritária deixando a proporção de pixels com ocorrências de MGMs (classe minoritária) igual a de sem ocorrências. O *train_test_split* realiza a separação dos dados em treino e teste, neste trabalho foi adotado uma separação de 80% para treino e 20% para teste (Zydrón et al., 2022).

A função definindoHiperparametros(), utiliza principalmente o GridSearchCV(), mas para a criação do modelo de MLP o MLPClassifier() é utilizado. O GridSearchCV() é uma ferramenta da biblioteca *scikit learn* que testa a combinação de hiper parâmetros de um modelo com base nos dados inseridos.

Os hiper parâmetros são características do modelo que podem ser selecionadas pelo usuário. A lista a seguir apresenta os hiper parâmetros que a função GridSearchCV() permite alterar para o *perceptron* multicamadas:

- Quantidade de camadas ocultas: intervalo de valores que define a quantidade de camadas ocultas.
- Quantidade de neurônios por camada oculta: intervalo de valores que define a quantidade de neurônios por camada oculta.
- Função de ativação: define a função de ativação no *perceptron*.
- Otimizadores: define o algoritmo que desempenha a melhor performance para atualizar os pesos do modelo.
- Taxa de aprendizado: define o tipo a taxa em que o modelo atualiza os pesos.

Neste trabalho os hiper parâmetros escolhidos foram quantidade de camadas ocultas, quantidade de neurônios em cada camada oculta, funções de ativação, otimizadores, e taxas de aprendizado, a tabela 4 resume todos os hiper parâmetros e os valores que foram testados.

Tabela 4: Metadados das imagens com as dimensões e os valores.

Hiper parâmetro	Valores
Quantidade de camadas ocultas	De 1 a 5
Quantidade de neurônios por camada oculta	De 10 a 40
Função de ativação	Relu
Otimizadores	lbfgs, sgd, adam
Taxa de aprendizado	Constante ou Adaptativo

Após o GridSearchCV() testar todas as combinações dos hiper parâmetros, é realizado um ranqueamento e a melhor combinação é utilizada para treinar o modelo. Seguindo Zydrón et al. (2022) adotou-se o "*f1-score*" (equação 15) como método de pontuação, esse método realiza a média harmônica da precisão (equação 13) e do "*recall*" (equação 14), que são métodos para avaliar o modelo utilizando a tabela 5 e os seguintes critérios:

1) Classe 0: pixel que foi classificado como não sendo a ocorrência de MGM.
2) Classe 1: pixel que foi classificado como sendo a ocorrência de MGM.

Tabela 5: Matriz de confusão

		Valor da Previsão	
		Sim	**Não**
Valor real	**Sim**	**Verdadeiro Positivo (True Positive - TP)** A previsão classificou o pixel como local suscetível à MGM e houve a ocorrência do mesmo	**Verdadeiro Negativo (True Negative - TN)** A previsão classificou o pixel como local suscetível à MGM, porém não houve a ocorrência do mesmo.
	Não	**Falso Positivo (False Positive - FP)** A previsão classificou o pixel como local não suscetível à MGM e houve a ocorrência do mesmo.	**Falso Negativo (False Negative - FN)** A previsão classificou o pixel como local não suscetível à MGM e não houve a ocorrência do mesmo.

$$precisão = \frac{TP}{TP + FP} \quad (Eq.13)$$

$$recall = \frac{TP}{TP + FN} \quad (Eq.14)$$

$$F1 - score = \frac{2 \times recall \times precisão}{recall + precisão} \quad (Eq.15)$$

O modelo com o melhor "f1" utilizou os parâmetros da Tabela 6 e obteve resultados apresentados na tabela 7.

Tabela 6: Metadados das imagens com as dimensões e os valores

Hiper parâmetro	Valores
Quantidade de camadas ocultas	5
Quantidade de neurônios por camada oculta	30
Função de ativação	Relu

Tabela 7: Relatório de classificação

Classes	Precisão	*Recall*	F1-score
0	0,87	1,0	0,93
1	1,00	0,86	0,92

4. RESULTADOS E DISCUSSÕES

Com os parâmetros utilizados obteve-se a Figura 26, o intervalo de probabilidade calculada pelo modelo apresentou valores entre 0 a 76%. Dentre os dados que foram inseridos

no modelo é possível conferir que os dados de topografia e ndvi tiveram grande influência na modelagem, com o dado de ndvi podendo ser responsável pelas áreas com probabilidades de 50-60% no centro do mapa.

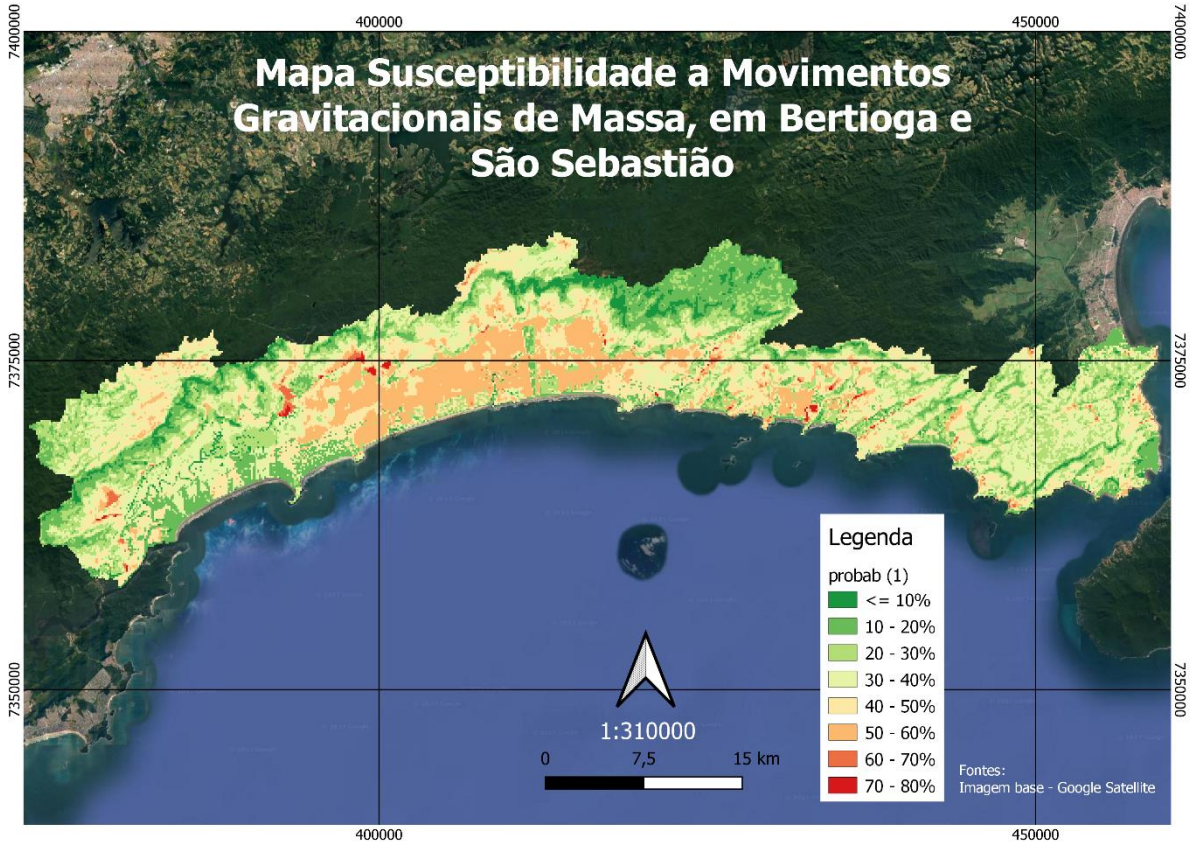

Figura 26: Imagem de Susceptibilidade

Porém, assim como visto anteriormente o MLP apresenta muitos parâmetros e dentre os parâmetros que a biblioteca scikit learn oferece existem os estados aleatórios (*random states*), esse parâmetro é um gerador de números aleatórios que ao receber um valor (*seed*) gera um valor aleatório para iniciar a ferramenta. Os locais em que esse parâmetro é utilizado são no *RandomUnderSampler*, *train_test_split* e no MLP, a presença desse parâmetro em três ferramentas adicionam aleatoriedade o que pode dificultar para treinar o modelo.

Outro ponto a ser considerado é sobre o histórico de deslizamentos, inicialmente este trabalho catalogaria os MGMs que ocorreram entre 2020 a 2022, utilizando as imagens de satélite (Sentinel 2) em cor falsa cor (SWIR1, NIR e Azul) para destacar a resposta do solo da vegetação. Porém não puderam ser observados nenhum evento, segundo o sistema integrado da defesa civil (SIDEC) houveram duas ocorrências no período analisado (Figura 27), elas não foram possíveis de observar por causa da resolução do pixel que tem 20 x 20 m. Por isso foi utilizado o evento que ocorreu em 19/02/2023, pois foi um evento com proporções que eram possíveis observar com a resolução do satélite.

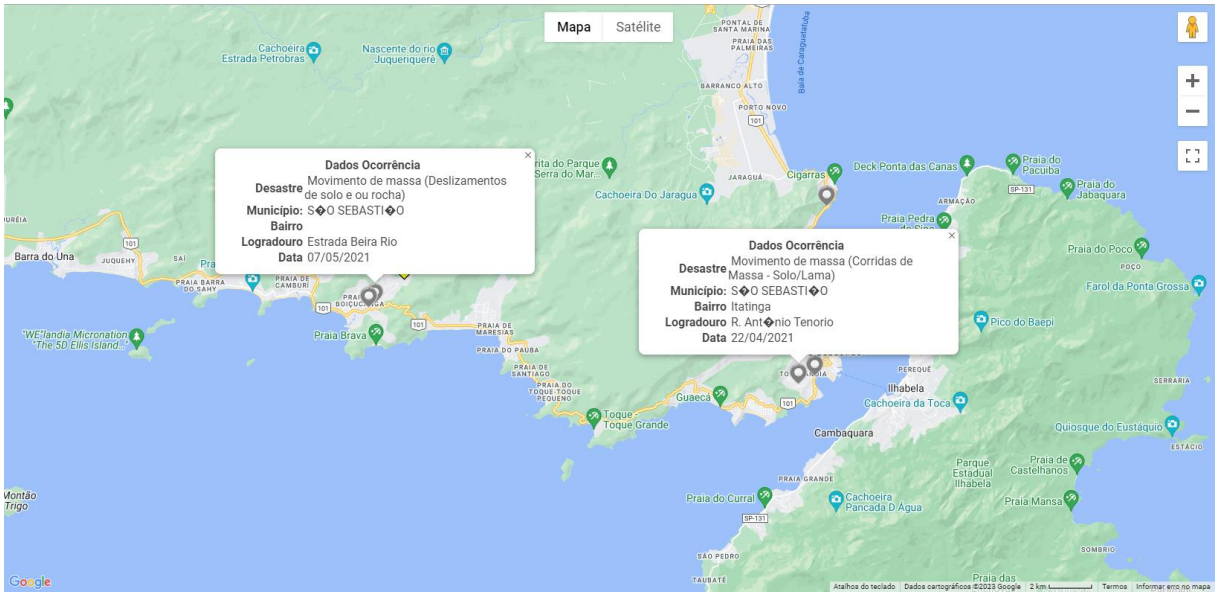

Figura 27: Registros de ocorrências de MGMs registrados pelo SIDEC

5. CONCLUSÕES

Neste trabalho a automatização da produção do mapa de susceptibilidade de Bertioga e São Sebastião foi realizada utilizando o método de MLP, essa ferramenta se mostrou útil para calcular a probabilidade de ocorrências de MGMs. Porém, devido a concentração de ocorrências de deslizamentos a modelagem ficou enviesada resultando em áreas com pouco declive apresentando alta probabilidade de ocorrer um MGM. A inserção de mais dados e dados mais esparsos devem melhorar o treinamento do modelo permitindo, assim, uma melhor acurácia desse.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Almeida, F.F.M. de, 2018, Fundamentos Geológicos do Relevo Paulista: Revista do Instituto Geológico, v. 39, no. 3, doi: 10.33958/revig.v39i3.600.

Antunes, F.S., Salomão, F.X.T., 2018, Solos em Pedologia, *in* Oliveira, A.M.S, Monticeli, J.J., Geologia de Engenharia e Ambiental: Associação de Geologia de Engenharia e Ambiental, p.72-85.

Bitar, O.Y., 2014, Cartas de suscetibilidade a movimentos gravitacionais de massa e inundações 1:25.000, São Paulo: IPT – Instituto de Pesquisas Tecnológicas, ISBN: 978-85-09-00177-3 (IPT).

Brito Neves, B. B. de, & Cordani, U. G. (1991). Tectonic evolution of south america during the late proterozoic. Precambrian Research, 53(1-2), 23-40. doi:10.1016/0301-9268(91)90004-T

Brito, M.C.W de., Oliveira, L.R.C.N. de., 2008, Plano de Manejo do Parque Estadual da Serra do Mar, São Paulo: IF – Instituto Florestal.

Dias Neto, C.M., 2001, Evolução tectono-termal do Complexo Costeiro (Faixa de Dobramentos Ribeira) em São Paulo [Tese de Doutorado]: São Paulo, Universidade de São Paulo, Instituto de Geosciências, doi: 10.11606/T.44.2001.tde-30092013-151641.

Goodfellow, I.J., Bengio, Y., Courville, A., 2016, Deep Learning: MIT Press.

Gonzalez, R.C., Woods, R.E., 2018, Digital Image Processing Global Edition: Pearson, 4º ed., 1022 p., ISBN 978-1-292-22304-9.

Guidicini, G., Nieble, C.M., 1984, Estabilidade de taludes naturais e de escavação,Editora Edgard Blucher, ISBN 978-85-212-0186-1.

Hairiah, K., Widianto, W., Suprayogo, D., Van Noordwijk, M., 2020, Tree Roots Anchoring and Binding Soil: Reducing Landslide Risk in Indonesian Agroforestry: Land, v. 9, i. 8, doi: https://doi.org/10.3390/land9080256.

Macedo, E.S. de, Sandre, L.H., 2022, Mortes por deslizamentos no Brasil: 1988 a 2022: Revista Brasileira de Geologia de Engenharia e Ambiental, v. 12, n. 1, p. 110-117.

Maxwell, J.C., 1865, A Dynamical Theory of the Electromagnetic Field: In Philosophical Transactions of the Royal Society of London, v. 155, p. 459-512, doi: https://doi.org/10.1098/rstl.1865.0008.

Meneses, P.R., Almeida, T., 2012, Introdução ao Processamento de Imagens de Sensoriamento Remoto: Brasília: CNPq, ed. 1, v. 1. 256p.

Morais, S.M., 1999, Integração Geológica da Folha Santos SF.23-Y-D: CPRM, 47 p.

Nalon, M.A., Matsukuma, C.K., Pavão, M., Ivanauskas, N.M., Kanashiro, M.M., 2022, Inventário da cobertura vegetal nativa do Estado de São Paulo, São Paulo: SIMA/IPA, ISBN 978-65-996417-2-5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V, Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011, Scikit-learn: Machine Learning in {P}ython: Journal of Machine Learning Research, v. 12, n. 85, p. 2825-2830.

Peñafiel P.G., Rojas, A.H., 2021, Landslide susceptibility index based on the integration of logistic regression and weights of evidence: A case study in Popayan, Colombia: Engineering Geology, v. 280, doi: https://doi.org/10.1016/j.enggeo.2020.105958.

Pimentel, J., Santos, T.D. dos, 2018, Manual de mapeamento de perigo e risco a movimentos gravitacionais de massa – Projeto de Fortalecimento da Estratégia Nacional de Gestão Integrada de Desastres Naturais – Projeto GIDES, Rio de Janeiro: CPRM/SGB, 213 p., ISBN: 978-85-7499-448-2.

Rosenblatt, F., 1958, The perceptron: A probabilistic model for information storage and organization in the brain: Psychological Review, v. 65, no. 6, p. 386–408. https://doi.org/10.1037/h0042519.

Rossi, M. 2017. Mapa pedológico do Estado de São Paulo: revisado e ampliado. São Paulo: Instituto Florestal, v. 1, p. 105-107, ISBN: 978-85-64808-16-4.

Rouse Junior, J. W., Haas, R. H., Schell, J. A., Deering, D. W., 1973, Monitoring the vernal advancement and retrogradation (green wave effect) of natural vegetation, Type II, 120 p.

Rumelhart, D., Hinton, G. & Williams, R., 1986, Learning representations by backpropagating errors: Nature, v. 323, p. 533–536, doi: https://doi.org/10.1038/323533a0.

Silva, A.P. de., Barroso, E.V., Polivanov, H., 2022, Índices pluviométricos críticos para prevenção de desastres por deslizamentos na cidade de Niterói, RJ: Geologia USP, Série científica, v. 22, n. 3, p. 47-60. doi: https://doi.org/10.11606/issn.2316-9095.v22-191653.

Stokes, A., Norris, J.E., Van Beek, L.P.H., Bogaard, T., Cammeraat, E., Mickovski, S.B., Jenner, A., Di Iorio, A., Fourcaud, T., 2008, How vegetation reinforces soil on slopes, in Norris, J.E, Stokes, A., Mickovski, S.B., Cammeraat, E., Beek, R., Nicoll, B.C., Achim, A., Slope Stability and Erosion Control: Ecotechnological Solutions: Springer: Dordrecht, The Netherlands, p. 65–118, https://doi.org/10.1007/978-1-4020-6676-4_4.

Tavares, R., Neto, J., Tommanselli, J., Pressinotti, M., Santoro, J., 2004, Análise da variabilidade temporal e espacial das chuvas associada aos movimentos de massa no litoral norte paulista, in Simpósio Brasileiro de Desastres Naturais, ed. 1, Anais do I SIBRADEN, p. 680-696.

Tavares, R., 2009, Clima, Tempo e desastres, in Tominaga, L.K., Santoro, J., Amaral, R. do, 2009, Desastres Naturais: conhecer para prevenir, Instituto Geológico, v. 1, ISBN 978-85-87235-09-1, p. 111-146.

Terzaghi, K., 1950, Mechanisms of Landslides, Geotechnical Society of America, Berkeley, p. 83-125.

Tominaga, L.K., Santoro, J., Amaral, R. do, 2009, Desastres Naturais: conhecer para prevenir, Instituto Geológico, v. 1, ISBN 978-85-87235-09-1.

Tominaga, L.K., Santoro, J., Amaral, R. do, 2015, Desastres Naturais: conhecer para prevenir, Instituto Geológico, v. 3, ISBN 978-85-87235-09-1.

Varnes, D.J., 1978, Slope Movement Types and Processes, in Schuster, R.L., Krizek, R.J., Eds., Landslides, Analysis and Control: Transportation Research Board, Special Report, National Academy of Sciences, no. 176, p. 11-33.

Wang, H., Zhang, L., Yin, K., Luo, H., Li, J., 2021, Landslide identification using machine learning, Geoscience Frontiers, v. 12, i. 1, p. 351-364, ISSN 1674-9871.

Wu, Q., 2020, geemap: A Python package for interactive mapping with Google Earth Engine: The Journal of Open Source Software, v. 5, n. 51, p. 2305. https://doi.org/10.21105/joss.02305.

Wyllie, D.C., 2017, Rock slope engineering: civil application, ed. 5, Editora CRC Press, ISBN 9781498786287.

Zydroń, T., Demczuk, P., Gruchot, A., 2022, Assessment of Landslide Susceptibility of the Wiśnickie Foothills Mts. (The Flysch Carpathians, Poland) Using Selected Machine Learning Algorithms: Frontiers in Earth Science, v. 10, doi:10.3389/feart.2022.872192.

7. ANEXOS

7.1. Anexo A

Código para o download das imagens disponível no GEE (https://colab.research.google.com/drive/1hhluH4WfFTqA564XUkQNecRSFvGtSJYy?usp=sharing)

```
#Instalação do geemap

#instalação da versão 0.16 do ipyleaflet por causa de um erro na visualização dos mapas do geemap
#https://github.com/gee-community/geemap/issues/1132
%pip install ipyleaflet==0.16

#Pacote necessário para a produção dos mapas
%pip install pygis

#Principal pacote que cria um mapa interativo
%pip install geemap

#Pacote para visualização de imagens
%pip install rasterio
#Importação

import ee #Pacote do earth engine

import pandas as pd #Importar o pacote de trabalho com tabelas
from IPython.display import display

import numpy as np #Importar o pacote de matemática

import os #Importar o pacote para trabalhar com diretórios

#Pacote para trabalhar com datas no python
```

```
from datetime import date
from datetime import timedelta
from datetime import datetime

#Importar as bibliotecas para plotar as imagens
import matplotlib as mpl
import matplotlib.pyplot as plt
import rasterio
from rasterio.plot import show
import geemap #Pacote para visualização do pacote ee
#Inicialização do geemap e gee
geemap.ee_initialize()
class Propriedades:

  #Função que inicializa a classe e abre a tabela de
propriedades
  def __init__(self, caminho):
    self.caminho_arquivo = caminho
    self.tabela_propriedades =
pd.read_csv(self.caminho_arquivo, index_col = 'propriedade')


  #Função para editar o arquivo de propriedades
  def editPropriedades(self):

    display(self.tabela_propriedades) #Apresentar a tabela
para facilitar a escolha da propriedade e edição dela

    propriedade_alter = input('\nQual das propriedades que
será alterada? ') #Questionamento para decidir qual
propriedade será editada

    #Loop para questionar novamente caso o que foi inserido
não seja uma das propriedades
    while propriedade_alter not in
propriedades.tabela_propriedades.index:
```

```
      print('\O que foi inserido ou foi digitado errado ou não
faz parte das propriedades')
      propriedade_alter = input('Por favor digite novamente:
')
      lista_propr_alter = [propriedade_alter]

    informacao_alter = input('\nQual a informação nova terá
esta propriedade? ') #Questionamento para decidir a nova
informação que a propriedade terá

    #Alterando o arquivo original para que fique salvo a nova
informação
    self.tabela_propriedades.loc[propriedade_alter]['informaca
o'] = informacao_alter
    self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')

    return self.tabela_propriedades
#Classe de para os dados de Google Earth Engine
class Colecao:

  #Função para inicializar a classe e valores
  def __init__(self,caminho_arq_propr):
    self.caminho_arquivo =
caminho_arq_propr
 #O caminho para o arquivo de propriedades
    self.tabela_propriedades =
Propriedades(caminho_arq_propr).tabela_propriedades        #A
tabela de propriedades
    self.produto =
self.tabela_propriedades.loc['selec_prod']['informacao']
        #O produto selecionado anteriormente
    self.datas =
self.tabela_propriedades.loc['selec_datas']['informacao']
          #O intervalo de datas selecionados anteriormente
```

```
    self.area_trabalho =
self.tabela_propriedades.loc['selec_area_trab']['informacao']
   #A área de trabalho selecionado anteriormente
    self.zoom =
self.tabela_propriedades.loc['zoom_desejado']['informacao']
            #O zoom escolhido anteriormente
    self.centro_x =
self.tabela_propriedades.loc['centro_coordenada_X']['informaca
o']     #A longitude para a visualização escolhida
anteriormente
    self.centro_y =
self.tabela_propriedades.loc['centro_coordenada_Y']['informaca
o']     #A latitude para a visualização escolhida
anteriormente
    self.codigo_produto =
self.tabela_propriedades.loc['codigo_produto']['informacao']
#O código do produto GEE escolhido anteriormente


  #Função que apresenta as bandas disponíveis
  def bandasDisponiveis(self):

    #listas com as bandas disponíveis no Sentinel 2
    lista_bandas = ['Aerossol', 'azul', 'verde',
'vermelho','Red Edge 1',
                    'Red Edge 2', 'Red Edge 3', 'Infravermelho
próximo',
                    'Red Edge 4','Vapor de água',
'Infravermelho de ondas curtas 1',
                    'Infravermelho de ondas curtas 2']

    #Criação de uma tabela em que os índices são as siglas
(B1, B2, B3 etc) e a coluna das bandas é composto pela lista
de bandas disponíveis
    self.tabela_bandas = pd.DataFrame(lista_bandas, index =
['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B8A', 'B9',
'B11', 'B12'])
```

```
    return self.tabela_bandas #Retorna a tabela de bandas
disponíveis


  #Função que devolve a lista de bandas utilizadas
anteriormente
  def listarBandas(self):
    if
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'ndvi':
      #Trazer a string das bandas escolhidas anteriormente
      bandas =
self.tabela_propriedades.loc['selec_bandas']['informacao']

      bandas_str = bandas[2:-2]
      lista_bandas = bandas_str.split("', '")

      self.lista_bandas =
lista_bandas                                          #Lista
das bandas utilizadas anteriormente

      return self.lista_bandas #Retorna a lista de bandas
utilizadas anteriormente

    else:
      #Trazer a string das bandas escolhidas anteriormente
      bandas =
self.tabela_propriedades.loc['selec_bandas']['informacao']

      self.lista_bandas =
bandas                                                #Lista
das bandas utilizadas anteriormente

      return self.lista_bandas #Retorna a lista de bandas
utilizadas anteriormente
```

```
  #Função para alterar o produto de ndvi, falsa cor ou terreno
  def editProduto(self):

    #Apresentando ao usuário qual foi o produto escolhido
anteriormente
    print('\nEste é o atual produto escolhido: ' +
str(self.produto))

    #Pergunta ao usuário para qual produto será alterado
    produto_alter = input('\nVocê deseja alterar para qual
produto?
(ndvi/falsa_cor/terreno/direcao_de_inclinacao_do_terreno/angul
o_de_inclinacao_do_terreno):\n')

    #Loop para respostas fora das opções
    while produto_alter not in
['ndvi','falsa_cor','terreno','direcao_de_inclinacao_do_terren
o','angulo_de_inclinacao_do_terreno']:
      print('\nHouve um erro de digitação')
      produto_alter = input('Por favor, digite novamente: \n')

    #Lista vazia para as bandas a serem selecionadas
    self.bandas = []

    #Condição caso o produto seja ndvi
    if produto_alter == 'ndvi':

      #Bloco que atualiza a tabela e algumas informações
      self.produto = produto_alter
      self.tabela_propriedades.loc['codigo_produto']['informac
ao'] = 'COPERNICUS/S2_SR_HARMONIZED'
      self.tabela_propriedades.loc['selec_prod']['informacao']
= produto_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
```

```
      #Bloco que altera as bandas para as que o ndvi utiliza e
atualiza a tabela
      self.bandas = ['B4','B8']
      self.tabela_propriedades.loc['selec_bandas']['informacao
'] = self.bandas
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')

      #Retorna as bandas escolhidas em forma de lista
      return self.bandas

    #Condição caso o produto seja falsa_cor
    elif produto_alter == 'falsa_cor':

      #Atualização da tabela com o produto sendo ndvi
      self.produto = produto_alter
      self.tabela_propriedades.loc['codigo_produto']['informac
ao'] = 'COPERNICUS/S2_SR_HARMONIZED'
      self.tabela_propriedades.loc['selec_prod']['informacao']
= produto_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')

      #Apresenta a tabela de bandas disponíveis
      display(self.bandasDisponiveis())

      #Loop para montar a composição de falsa cor
      for falsa_cor in ['vermelho', 'verde', 'azul']:

        #Pergunta ao usuário para escolher a banda que
assumirá uma das posições escolhidas no momento
        banda = input('Digite a banda que gostaria que
assumisse a posição do ' + falsa_cor + ' na composição: ')

        #Loop para respostas fora da lista de bandas
disponíveis
        while banda not in self.tabela_bandas.index:
```

```
          print('Houve um erro na digitação.')
          banda = input('Por favor, digite novamente: ')

        #Adição das bandas na lista
        self.bandas.append(banda)

      #Atualização da tabela
      self.tabela_propriedades.loc['selec_bandas']['informacao
'] = self.bandas
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')

      return self.bandas #Retorna as bandas escolhidas em
forma de lista

    #Condição para caso terreno seja escolhida
    else:
      #Bloco que atualiza a tabela e as informações do produto
      self.produto = produto_alter
      self.tabela_propriedades.loc['codigo_produto']['informac
ao'] = 'JAXA/ALOS/AW3D30/V3_2'
      self.tabela_propriedades.loc['selec_prod']['informacao']
= produto_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')


  #Função para mudar o intervalo de data
  def editIntervaloDatas(self):

    #Bloco que recebe as datas utilizadas anteriormente e
informa ao usuário
    self.data_inicial =
self.tabela_propriedades.loc['data_inicial']['informacao']
    self.data_final =
self.tabela_propriedades.loc['data_final']['informacao']
    self.datas = [self.data_inicial, self.data_final]
```

```
    print('Este é o atual intervalo de data ' +
str(self.datas))

    #Linha para questionar qual das datas devem ser alteradas
    data_alter = input('\nVocê deseja alterar a data inicial,
final ou ambas? (inicial/final/ambas)')

    #Loop para respostas fora do sugerido
    while data_alter not in ['inicial', 'final','ambas']:
      print('\nHouve um erro na digitação')
      data_alter = input('Por favor, digite novamente: ')

    #Condição para alterar a data inicial
    if data_alter == 'inicial':
      #Bloco para alterar a data inicial e atualizar na tabela
de propriedades
      nova_data = input('Digite a nova data no formato YYYY-
MM-DD: ')
      self.data_inicial = nova_data
      self.tabela_propriedades.loc['data_inicial']['informacao
'] = nova_data
      self.datas = [self.data_inicial, self.data_final]
      self.tabela_propriedades.loc['selec_datas']['informacao'
] = self.datas
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      return print('O novo intervalo é de ' + str(self.datas))

    #Condição para alterar a data inicial
    elif data_alter == 'final':
      #Bloco para alterar a data final e atualizar na tabela
de propriedades
      nova_data = input('Digite a nova data no formato YYYY-
MM-DD: ')
      self.data_final = nova_data
      self.tabela_propriedades.loc['data_final']['informacao']
= nova_data
```

```
      self.datas = [self.data_inicial, self.data_final]
      self.tabela_propriedades.loc['selec_datas']['informacao'
] = self.datas
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      return print('O novo intervalo é de ' + str(self.datas))

    #Condição para alterar ambas as datas
    else:
      nova_data_inicial = input('Digite a nova data inicial no
formato YYYY-MM-DD: ')
      self.data_inicial = nova_data_inicial
      self.tabela_propriedades.loc['data_inicial']['informacao
'] = nova_data_inicial
      nova_data_final = input('Digite a nova data final no
formato YYYY-MM-DD: ')
      self.data_final = nova_data_final
      self.tabela_propriedades.loc['data_final']['informacao']
= nova_data_final
      self.datas = [self.data_inicial, self.data_final]
      self.tabela_propriedades.loc['selec_datas']['informacao'
] = self.datas
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      return print('O novo intervalo é de ' + str(self.datas))

  #def listarDatasPassagem(self):



  #Função para alterara área de trabalho
  def editAreaTrabalho(self):

    #Apresentando a área de trabalho que foi utilizado
anteriormente
    print('A área de trabalho atual é: ' +
str(self.area_trabalho))
```

```
    #Perguntando ao usuário para qual área de trabalho será
alterado
    area_alter = input('Deseja alterar par qual área de
trabalho: (Bertioga/Sao_Sebastiao/Total/Treino)')

    #Loop para respostas fora do sugerido
    while area_alter not in ['Bertioga', 'São_Sebastião',
'Total', 'Treino']:
      print('\nHouve um erro na digitação')
      area_alter = input('Por favor, digite novamente: ')

    #Condição para área de trabalho sendo Bertioga, alterando
e atualizando a tabela de propriedades
    if area_alter == 'Bertioga':
      self.tabela_propriedades.loc['selec_area_trab']['informa
cao'] = area_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      self.transfAreaTrabalho()
                  #Tranformação do caminho da área de trabalho
para uma classe de método geometry do GEE
      return
area_alter
    #Retorna a área de trabalho escolhida como um método do
GEE

    #Condição para área de trabalho sendo São Sebastião,
alterando e atualizando a tabela de propriedades
    elif area_alter == 'Sao_Sebastiao':
      self.tabela_propriedades.loc['selec_area_trab']['informa
cao'] = area_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      self.transfAreaTrabalho()
                  #Tranformação do caminho da área de trabalho
para uma classe de método geometry do Google earth Engine
```

```
      return
area_alter
    #Retorna a área de trabalho escolhida como um método do
GEE

    #Condição para área de trabalho sendo total, alterando e
atualizando a tabela de propriedades
    elif area_alter == 'Total':
      self.tabela_propriedades.loc['selec_area_trab']['informa
cao'] = area_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      self.transfAreaTrabalho()
                  #Tranformação do caminho da área de trabalho
para uma classe de método geometry do Google earth Engine
      return
area_alter
    #Retorna a área de trabalho escolhida como um método do
GEE

    #Condição para área de trabalho sendo Treino, alterando e
atualizando a tabela de propriedades
    else:
      self.tabela_propriedades.loc['selec_area_trab']['informa
cao'] = area_alter
      self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')
      self.transfAreaTrabalho()
                  #Tranformação do caminho da área de trabalho
para uma classe de método geometry do Google earth Engine
      return
area_alter
    #Retorna a área de trabalho escolhida como um método do
GEE


  #Função que transforma a área de trabalho escolhida para
```

```
  def transfAreaTrabalho(self):

    #Dicionário para as respostas esperadas do usuário e
índice correspondente ao caminho do arquivo kml da área de
trabalho
    dicion_area_trabalho = {'Bertioga':'caminho_at_bertioga',
                            'São_Sebastião':'caminho_at_saoseb
ast',
                            'Total':'caminho_area_trabalho',
                            'Treino':'caminho_at_treino'}

    #Bloco que transforma o arquivo kml para uma geometria do
Earth Explorer e logo em seguida em um método geometry
    ee_geom =
geemap.kml_to_ee(self.tabela_propriedades.loc[dicion_area_trab
alho[self.area_trabalho]]['informacao'])  #Transformação da
área de trabalho no tipo ee.Geometry
    ee_geom_met =
ee_geom.geometry()
                                      #Transformação da área de
trabalho no médoto geometry
    self.area_geom = ee_geom
    self.area_metodo = ee_geom_met
    return self.area_metodo #Retorna o método da área de
trabalho


  #Função que apresenta as informações da coleção em forma de
tabela
  def verInfoColecao(self):
    #Transformando as informações de produto, data e área de
trabalho em uma série do pandas
    lista_info_colecao = [self.produto, self.datas,
self.area_trabalho]
    #Criação da tabela de informações da coleção
    self.tabela_info_colecao =
pd.DataFrame(lista_info_colecao, index=['produto',
```

```
        'intervalo de datas',

        'area de trabalho'])
    return self.tabela_info_colecao #Retorna a tabela de
coleção


  #Função para editar as informações da coleção
  def editInfoColecao(self):

    #Mostrando a tabela de informações da coleção
    display(self.verInfoColecao())

    #Perguntando ao usuário qual a propriedade que se gostaria
de alterar
    info_colec = input('Qual das propriedades gostaria de
alterar? (produto/intervalo_de_datas/area_de_trabalho)')

    #Loop para respostas fora do sugerido
    while info_colec not in ['produto', 'intervalo_de_datas',
'area_de_trabalho']:
      print('\nHouve um erro na digitação')
      info_colec = input('Por favor, digite novamente: ')

    #Condição para alterar o produto
    if info_colec == 'produto':
      self.editProduto()

    #Condição para alterar o intervalo de datas
    elif info_colec == 'intervalo_de_datas':
      self.editIntervaloDatas()

    #Condição para alterar a área de trabalho
    else:
      self.editAreaTrabalho()
```

```
  #Função apra montar a coleção GEE
  def montarColecao(self):

    #Pegando a data inicial e a final da tabela de
propriedades
    data_inicial =
self.tabela_propriedades.loc['data_inicial']['informacao']
    data_final =
self.tabela_propriedades.loc['data_final']['informacao']


    if
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'ndvi' or
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'falsa_cor':
    #Bloco que cria a coleção de imagens do GEE utilizando as
informações de produto, intervalo de data, bandas escolhidas e
área de trabalho
      self.colecao = (
          ee.ImageCollection(self.codigo_produto)
                    #Código do produto GEE
          .filterDate(data_inicial,
data_final)                                    #Intervalo de
datas
          .select(self.listarBandas())
                   #Bandas escolhidas
          .filterBounds(self.transfAreaTrabalho())
                   #Área de trabalho no formato de método
geometry
          )

      return self.colecao  #Retorna a coleção no formato
ee.imageCollection

    else:
      self.colecao = (
```

```
          ee.ImageCollection(self.codigo_produto)
                    #Código do produto GEE
          .filterBounds(self.transfAreaTrabalho())
                   #Área de trabalho no formato de método
geometry
          )

      return self.colecao  #Retorna a coleção no formato
ee.imageCollection

  #Função que transforma a coleção de imagens em uma imagem e
recorta
  def montarImagem(self):

    #Condição para montar a imagem caso o produto escolhido
seja falsa cor
    if
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'falsa_cor':
      colecao =
self.montarColecao()
          #Montando a coleção
      colecao_clip = colecao.map(lambda image:
image.clip(self.transfAreaTrabalho()))    #Recortando a imagem
      imagem_clip =
colecao_clip.median()
       #Transformando a coleção de imagem em uma imagem
utilizando uma mediana

      return imagem_clip #Retorna a imagem em falsa cor
recortada

    #Condição para montar a imagem caso o produto escolhido
seja ndvi
    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'ndvi':
```

```
      #Bloco para montar a coleção, recortar e criar a imagem
      colecao = self.montarColecao()
      colecao_clip = colecao.map(lambda image:
image.clip(self.transfAreaTrabalho()))
      imagem_clip = colecao_clip.median()

      #Bloco para pegar as bandas do vermelho e NIR para
aplicar o cálculo do ndvi
      lista_bandas = self.listarBandas()
      nir = imagem_clip.select(lista_bandas[1])
      verm = imagem_clip.select(lista_bandas[0])
      ndvi = nir.subtract(verm).divide(nir.add(verm))

      return ndvi #Retorna a imagem ndvi

    #Condição para montar a imagem caso o produto escolhido
seja terreno
    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'terreno':
      #Bloco para montar a coleção, recortar e criar a imagem
      colecao = self.montarColecao()
      colecao_clip = colecao.map(lambda image:
image.clip(self.transfAreaTrabalho()))
      colecao_clip = colecao_clip.median()
      elevacao_clip = colecao_clip.select('DSM')

      return elevacao_clip #Retorna a imagem de terreno
recortada

    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'direcao_de_inclinacao_do_terreno':
      #Bloco para montar a coleção, recortar e criar a imagem
      colecao = self.montarColecao()
      colecao_clip = colecao.map(lambda image:
image.clip(self.transfAreaTrabalho()))
```

```
      colecao_clip = colecao_clip.median()
      elevacao_clip = colecao_clip.select('DSM')
      elevacao_clip.rename('elevation')
      aspect = ee.Terrain.aspect(elevacao_clip)

      return aspect #Retorna a imagem de terreno recortada

    else:
      #Bloco para montar a coleção, recortar e criar a imagem
      colecao = self.montarColecao()
      colecao_clip = colecao.map(lambda image:
image.clip(self.transfAreaTrabalho()))
      colecao_clip = colecao_clip.median()
      elevacao_clip = colecao_clip.select('DSM')
      elevacao_clip.rename('elevation')
      slope = ee.Terrain.slope(elevacao_clip)

      return slope #Retorna a imagem de terreno recortada


  #Função para a visualização das imagens
  def visualizacaoImagem(self):

    #Montando o mapa para a visualização da imagem
    mapa_ee = geemap.Map(center =
(float(self.centro_x),                          #Longitude em
que estará o centro da visualização
                                    float(self.centro_y)),
                  #Longitude em que estará o centro da
visualização
                          zoom =
int(self.zoom))                                  #Zoom em que
estará a visualização

    #Montando a imagem
    imagem = self.montarImagem()
```

```
    #Para cada tipo de imagem há uma visualização diferente
      #Definindo os parâmetros para a imagem em falsa cor
    if
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'falsa_cor':

      lista_bandas = self.listarBandas()

      parametros_vis = {
          'min' : 0.0,
          'max' : 6000,
          'bands' : lista_bandas
          }

      #Definindo os parâmetros para a imagem ndvi
    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'ndvi':
      parametros_vis = {
          'palette': [
              'FFFFFF', 'CE7E45', 'DF923D', 'F1B555',
'FCD163', '99B718',
              '74A901','66A000', '529400', '3E8601', '207401',
'056201',
              '004C00', '023B01','012E01', '011D01', '011301'
              ]
          }


    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'terreno':
      parametros_vis = {
          'min': 0,
          'max': 5000,
          'palette': ['0000ff', '00ffff', 'ffff00', 'ff0000',
'ffffff']
```

```
          }

    elif
self.tabela_propriedades.loc['selec_prod']['informacao'] ==
'direcao_de_inclinacao_do_terreno':
      parametros_vis = {min: 0, max: 359.99}

    else:
      parametros_vis = {min: 0, max: 89.99}

    #Adicionando a camada de imagem para o mapa de
visualização
    mapa_ee.addLayer(imagem, parametros_vis,'Layer para
conferir')

    return mapa_ee #Retorna o mapa interativo com a imagem


  #FUNÇÃO PARA BAIXAR AS IMAGENS
  def baixarImagem(self):

    #Caminho para cada tipo arquivo
    caminhos_download = {
        'ndvi':self.tabela_propriedades.loc['caminho_ndvi']['i
nformacao'],
        'falsa_cor':self.tabela_propriedades.loc['caminho_cor_
falsa']['informacao'],
        'terreno':self.tabela_propriedades.loc['caminho_mde'][
'informacao'],
        'direcao_de_inclinacao_do_terreno':self.tabela_proprie
dades.loc['caminho_aspect']['informacao'],
        'angulo_de_inclinacao_do_terreno':self.tabela_propried
ades.loc['caminho_slope']['informacao']
                          }

    #Nome para o arquivo
```

```
    nome =
(self.tabela_propriedades.loc['data_inicial']['informacao'] +
'_' + self.tabela_propriedades.loc['selec_prod']['informacao']
+ '.tif')

    #Montando o caminho para o arquivo
    caminho_arq =
os.path.join(caminhos_download[self.tabela_propriedades.loc['s
elec_prod']['informacao']],
                                nome)

    #Montando a imagem
    imagem = self.montarImagem()

    #Escala
    escala = {
        'ndvi':125,
        'falsa_cor':20,
        'terreno':30,
        'direcao_de_inclinacao_do_terreno':30,
        'angulo_de_inclinacao_do_terreno':30
                        }

    #Baixar a imagem
    geemap.download_ee_image(imagem,
                             region =
self.transfAreaTrabalho(),
                             filename = caminho_arq,
                             crs = 'EPSG:31983',
                             scale =
escala[self.tabela_propriedades.loc['selec_prod']['informacao'
]])
```

7.2. Anexo B

Código para a produção do mapa de susceptibilidade (https://colab.research.google.com/drive/1balvrlg54DxpOPoiq_hbsc6oM4Q2bpz6?usp=sharing)

```
#Instalação das bibliotecas

#Instalação das bibliotecas para trabalhar com imagens e
informações espaciais
%pip install rasterio
%pip install geopandas

#Instalação da biblioteca que adiciona uma barra de cores aos
plots
%pip install matplotlib-scalebar

#Instalação das bibliotecas de inteface de usuário
!pip install ipywidgets
# To enable `ipywidgets`
!jupyter nbextension enable --py widgetsnbextension
# Importar os pacotes

import os                                                    #
Biblioteca para trabalhar com o sistema operacional (no caso
google drive)
import numpy as np                                           #
Biblioteca para trabalhar com os dados em forma de matrizes
import pandas as pd                                          #
Biblioteca para trabalhar com tabelas

# Bibliotecas para trabalhar com a plotagem das imagens
from numpy import newaxis
import matplotlib as mpl
import matplotlib.pyplot as plt
from IPython.display import display
from matplotlib_scalebar.scalebar import ScaleBar

#import geopandas as gpd
# Bibliotecas para trabalhar com imagens
import rasterio
from rasterio.enums import Resampling
```

```
# Bibliotecas para trabalhar com o Machine Learning
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
class Propriedades:

  #Função que inicializa a classe e abre a tabela de
propriedades
  def __init__(self, caminho):
    self.caminho_arquivo = caminho
    self.tabela_propriedades =
pd.read_csv(self.caminho_arquivo, index_col = 'propriedade')


  #Função para editar o arquivo de propriedades
  def editPropriedades(self):

    display(self.tabela_propriedades) #Apresentar a tabela
para facilitar a escolha da propriedade e edição dela

    propriedade_alter = input('\nQual das propriedades que
será alterada? ') #Questionamento para decidir qual
propriedade será editada

    #Loop para questionar novamente caso o que foi inserido
não seja uma das propriedades
    while propriedade_alter not in
propriedades.tabela_propriedades.index:
      print('\O que foi inserido ou foi digitado errado ou não
faz parte das propriedades')
      propriedade_alter = input('Por favor digite novamente:
')
      lista_propr_alter = [propriedade_alter]
```

```
    informacao_alter = input('\nQual a informação nova terá
esta propriedade? ') #Questionamento para decidir a nova
informação que a propriedade terá

    #Alterando o arquivo original para que fique salvo a nova
informação
    self.tabela_propriedades.loc[propriedade_alter]['informaca
o'] = informacao_alter
    self.tabela_propriedades.to_csv(self.caminho_arquivo,
mode='w')

    return self.tabela_propriedades
class Imagens:

  #Função de inicialização
  def __init__(self, caminho):
    self.caminho_propriedades =
caminho
                          #O caminho para o arquivo de
propriedades
    self.tabela_propriedades =
Propriedades(caminho).tabela_propriedades
                           #A tabela de propriedades
    self.escolhendoTreinoTotal()
                                                        #Cham
ando a função que irá selecionar se os dados trabalhados serão
utilizados
    self.geotiffs = [os.path.join(self.caminho_escolhido,x)
for x in os.listdir(self.caminho_escolhido) if '.tif' in x]
#Linha que lê e lista os arquivos .tif que estão na pasta
    self.tabelaDimensoesImagens()
                                                       #Cha
ma a função que cria a tabela com as dimensões das imagens
```

```
  #Função que pergunta qual área quer utilizar, treino ou
total
  def escolhendoTreinoTotal(self):

    #Pergunta e dá as escolhas
    self.esc_treino_total = input('Qual será o tipo de imagem?
(treino/total) ')

    #Loop caso a entrada seja diferente do sugerido
    while self.esc_treino_total not in ['treino','total']:
      self.esc_treino_total = input('Parece que houve um erro
de digitação, tente novamente: (treino/total) ')

    #Condições para a escolha
    if self.esc_treino_total == 'treino':
      self.caminho_escolhido =
self.tabela_propriedades.loc['caminho_dados_treino']['informac
ao']
    else:
      self.caminho_escolhido =
self.tabela_propriedades.loc['caminho_dados_total']['informaca
o']


  #Função para descobrir o fator que será utilizado para
redimensionar as imagens
  def fatorRedimensionar(self):

    #Listas vazias para armazenar os valores de altura e
largura das imagens
    lista_fator_larg, lista_fator_altu = [], []

    #Encontrar as dimensões das menores imagens
    menor_larg = self.tabela_imagens.min()['largura']
    menor_altu = self.tabela_imagens.min()['altura']
```

```
    #Loop para encontrar o valor que será utilizado para
diminuir o tamanho o pixel,
    #a partir da quantidade de pixels atual e da quantidade de
pixels da menor imagem
    for indice in self.tabela_imagens.index:

      #Encontrar as dimensões da imagem
      valor_larg = self.tabela_imagens.loc[indice]['largura']
      valor_altu = self.tabela_imagens.loc[indice]['altura']

      #Cálculo do fator
      fator_larg = menor_larg/valor_larg
      fator_altu = menor_altu/valor_altu

      #Adicionar o fator na lista
      lista_fator_larg.append(fator_larg)
      lista_fator_altu.append(fator_altu)

    #Adicionando as colunas dos fatores na tabela de dimensões
    self.tabela_imagens['fator_largura'] = lista_fator_larg
    self.tabela_imagens['fator_altura'] = lista_fator_altu

    return self.tabela_imagens


  #Função para conferir a proporção das imagens e preparar uma
tabela com essas informações
  def tabelaDimensoesImagens(self):

    #Listas vazias para ir montando as colunas da tabela de
dimensões
    largura, altura, bandas, nome_arq, proporcao,
caminho_imagem = [], [], [], [], [], []

    #Loop para passar por todos as imagens
    for imagem in self.geotiffs:
```

```
      with rasterio.open(imagem,'r') as src: #Linha para abrir
a imagem como leitura essa ser chamada de src

        #Bloco para ler os dados e transformar em um ndarray
        imagem_np_array =
src.read(1)                          #Fazer leitura da primeira
banda que tem as informações,

#já que algumas imagens tem a segunda banda como alpha

        mascara_sem_dados = imagem_np_array
<0                #Linha que armazena os valores menores que
zero

        imagem_np_array[mascara_sem_dados] =
np.nan           #Linha que transforma os valores menores que
zero em np.nan

        #Linha que trabalha pega os nomes dos arquivos
        nome_arq.append(os.path.basename(imagem).replace('.tif
','''))

        #Bloco que armazena os dados de caminho da imagem,
dimensões, proporção e títulos
        caminho_imagem.append(imagem)

        largura.append(imagem_np_array.shape[1])

        altura.append(imagem_np_array.shape[0])

        proporcao.append(imagem_np_array.shape[1]/imagem_np_ar
ray.shape[0])

    #Linha que cria a tabela com dos dados das imagens
    self.tabela_imagens = pd.DataFrame({'largura':largura,
'altura':altura,
                                        'proporção':proporcao,
```

```
                                          'caminho_imagem':camin
ho_imagem},
                                        index=nome_arq)

    #Atualização da tabela com o fatores que vão ser
utilizados para redimensionalizar
    self.fatorRedimensionar()

    return self.tabela_imagens #Retorna a tabela com as
dimensões das imagens e o fator para redimensionar elas


  #Função para redimensionar as iamgens
  def redimensionarImagem(self, indice):


    geotif = self.tabela_imagens.loc[indice]['caminho_imagem']

    lista_dados_categoricos =
['Img_MascBin_Treino','Img_Geologia_Treino',
                                'Img_Pedologia_Prof_Treino',
'Img_Pedologia_Treino',
                                'Img_Vegetacao_Treino','Img_Geo
logia_Total',
                                'Img_Pedologia_Prof_Total',
'Img_Pedologia_Total',
                                'Img_Vegetacao_Total']

    with rasterio.open(geotif) as dataset:

      if indice in lista_dados_categoricos:

        #Redimensionar o tamanho da imagem utilizando nearest
neighbor
        dataset_dim = dataset.read(
            out_shape = (
                dataset.count,
```

```
                int(round(dataset.height *
self.tabela_imagens.loc[indice]['fator_altura'])),
                int(round(dataset.width *
self.tabela_imagens.loc[indice]['fator_largura']))
                ),
            resampling=Resampling.nearest
            )

        return dataset_dim

      else:
        #Redimensionar o tamanho da imagem utilizando bilinear
        dataset_dim = dataset.read(
            out_shape = (
                dataset.count,
                int(round(dataset.height *
self.tabela_imagens.loc[indice]['fator_altura'])),
                int(round(dataset.width *
self.tabela_imagens.loc[indice]['fator_largura']))
                ),
            resampling=Resampling.bilinear
            )

        return dataset_dim


  #Função que retira as informações espaciais
  def informEspaciais(self):
    tabela = self.tabela_imagens
    menor_larg = tabela.min()['largura']
    menor_altu = tabela.min()['altura']
    self.menor_dim = tabela.loc[(tabela['largura'] ==
menor_larg) & (tabela['altura'] == menor_altu)]

    geotif =
self.tabela_imagens.loc[self.menor_dim.index[0]]['caminho_imag
em']
```

```
    with rasterio.open(geotif, 'r') as src:
      self.transform = src.transform
      self.limites = (src.bounds.left,
                      src.bounds.right,
                      src.bounds.bottom,
                      src.bounds.top)
      self.crs = src.crs

    return


  #Lendo os dados dos geotiffs
  def prepararImagens(self):

    self.fatorRedimensionar()

    caminho = self.caminho_escolhido

    dados, nomes = [], []

    for indice in self.tabela_imagens.index:

      dataset = self.redimensionarImagem(indice)

      #Bloco para transformar os valores que não devem ser
contados em np.nan
      mascara_sem_dados = dataset < 0
      dataset[mascara_sem_dados] = np.nan
      dados.append(dataset[0])
                  #Assim que tirar os dados sem valores,
adicionar na lista apenas as bandas com valores, já que
algumas imagens apresentam a banda alpha

      nome_arq = caminho + '/' + indice + '.npy'
      np.save(nome_arq, dados)
```

```
    self.np_stack = np.stack(dados) #Criar uma array 3D no
numpy

    return self.np_stack

  def atualizarTabelaValMinMax(self):
    self.prepararImagens()
    lista_valores_min = []
    lista_valores_max = []
    for array in self.np_stack:
      lista_valores_min.append(np.nanmin(array))
      lista_valores_max.append(np.nanmax(array))

    self.tabela_imagens['valor_min'] = lista_valores_min
    self.tabela_imagens['valor_max'] = lista_valores_max

    return
#Classe para trabalhar com o Machine Learning
class MLP:


  #Função de inicialização
  def __init__(self, caminho):
    self.caminho_propriedades =
caminho                                          #O caminho
para o arquivo de propriedades
    self.tabela_propriedades =
Propriedades(caminho).tabela_propriedades        #A tabela de
propriedades
    self.imagens = Imagens(caminho)
    self.tabela_imagens = self.imagens.tabela_imagens
    self.caminho_treino =
self.tabela_propriedades.loc['caminho_dados_treino']['informac
ao']
    self.caminho_total =
self.tabela_propriedades.loc['caminho_dados_total']['informaca
o']
```

```
    self.empilharArrays()


  #Função que empilha os np arrays dependendo da escolha de
total ou treino
  def empilharArrays(self):
    self.nparrays_treino =
[os.path.join(self.caminho_treino,x) for x in
os.listdir(self.caminho_treino) if '.npy' in x]
    self.nparrays_total = [os.path.join(self.caminho_total,x)
for x in os.listdir(self.caminho_total) if '.npy' in x]

    for array in self.nparrays_treino:
      self.np_stack_treino = np.load(array)

    for array in self.nparrays_total:
      self.np_stack_total = np.load(array)

    return


  #Função que normaliza os dados
  def normalizarDados(self):
    dados_norm = []
    dados_tot_norm = []

    for array in self.np_stack_treino:
      array_norm = (array-np.nanmin(array))/(np.nanmax(array)-
np.nanmin(array))
      dados_norm.append(array_norm)

    self.np_stack_treino_norm = np.stack(dados_norm)


    for array_total in self.np_stack_total:
```

```
      array_total_norm = (array_total-
np.nanmin(array_total))/(np.nanmax(array_total)-
np.nanmin(array_total))
      dados_tot_norm.append(array_total_norm)

    self.np_stack_total_norm = np.stack(dados_tot_norm)

    return


  #Função que cria um array de "sem dados" (np.nan)
  def mascaraSemDados(self):

    mascara = np.isnan(self.np_stack_treino[0])
    for imagem in self.np_stack_treino[1:]:
      mascara += np.isnan(imagem)


    mascara_total = np.isnan(self.np_stack_total[0])
    for imagem in self.np_stack_total:
      mascara_total += np.isnan(imagem)

    self.mascara_sem_dados = mascara
    self.mascara_sem_dados_total = mascara_total

    return


  #Função que prepara os dados, transformando os dados 2d em
1d e retirando os valores de np.nan
  def preparandoDados(self):
    self.normalizarDados()
    self.mascaraSemDados()
    entrada_np_array = self.np_stack_treino_norm[1:]
    etiqueta_np_array = self.np_stack_treino_norm[0]
    mascara = self.mascara_sem_dados.flatten()
    total_np_array = self.np_stack_total_norm
```

```
    mascara_total = self.mascara_sem_dados_total.flatten()

    self.entrada_pix =
entrada_np_array.reshape(entrada_np_array.shape[0],entrada_np_
array.shape[1]*entrada_np_array.shape[2]).T
    self.etiqueta_pix = etiqueta_np_array.flatten()
    self.total_pix =
total_np_array.reshape(total_np_array.shape[0],total_np_array.
shape[1]*total_np_array.shape[2]).T

    entrada = self.entrada_pix[~mascara]
    etiqueta = self.etiqueta_pix[~mascara]
    total = self.total_pix[~mascara_total]

    self.entrada = entrada
    self.etiqueta = etiqueta
    self.total = total

    return


  #Função que separa a área de treino em treino e teste com
70% para o treino e 30% para testar
  def separandoTreinoTeste(self):

    self.preparandoDados()

    rus = RandomUnderSampler(random_state=30)
    entrada_amost, etiqueta_amost =
rus.fit_resample(self.entrada, self.etiqueta)

    entrada_treino, entrada_teste, etiqueta_treino,
etiqueta_teste = train_test_split(entrada_amost,
etiqueta_amost, test_size=0.3, random_state=26)

    self.entrada_treino = entrada_treino
    self.entrada_teste = entrada_teste
```

```
    self.etiqueta_treino = etiqueta_treino
    self.etiqueta_teste = etiqueta_teste

    return


  #Função para criar o modelo
  def criandoModelo(self):

    mlp = MLPClassifier(random_state = 10,
                        shuffle = True)
    self.modelo = mlp

    return


  #Função para definir os hiperparâmetros e treinar o modelo
  def definindoHiperparametros(self):
    self.separandoTreinoTeste()
    self.criandoModelo()

    parametros = {
        'hidden_layer_sizes': [(10,1),(20,1),(30,1),(40,1),
                               (10,2),(20,2),(30,2),(40,2),
                               (10,3),(20,3),(30,3),(40,3),
                               (10,4),(20,4),(30,4),(40,4),
                               (10,5),(20,5),(30,5),(40,5)],
        'activation': ['relu'],
        'solver': ['lbfgs','sgd','adam'],
        'learning_rate': ['constant','adaptive'],
        }

    GS = GridSearchCV(
        estimator = self.modelo,
        param_grid = parametros,
        scoring =
['accuracy','average_precision','recall','f1'],
```

```
        refit = 'f1',
        verbose = 4
        )

    GS.fit(self.entrada_treino, self.etiqueta_treino)

    self.modelo_atualiz = GS
    self.modelo_param = GS.best_estimator_

    return


  #Função que pega o modelo e testa na área total grando o
mapa de probabilidade
  def criandoImagemProbab(self):
    self.mascaraSemDados()
    self.preparandoDados()
    self.preparandoDados()
    self.definindoHiperparametros()

    self.entrada =
self.total_pix[np.invert(self.mascara_sem_dados_total.flatten(
))]
    self.probab =
self.modelo_atualiz.predict_proba(self.entrada)[:,1]
    array_probab =
np.zeros(shape=self.mascara_sem_dados_total.flatten().shape,
dtype='float32')
    array_probab[np.invert(self.mascara_sem_dados_total.flatte
n())] = self.probab
    array_probab =
array_probab.reshape(self.mascara_sem_dados_total.shape)
    array_probab[self.mascara_sem_dados_total] = np.nan

    self.array_probab = array_probab

    return plt.imshow(array_probab)
```

```
  #Função gera o relatório da validação f1-score
  def montandoRelatorio(self):
    self.separandoTreinoTeste()

    self.previsao =
self.modelo_atualiz.predict(self.entrada_teste)
    print(classification_report(self.etiqueta_teste,
self.previsao))


  #Função que baixa a imagem de probabilidade
  def baixarImagem(self):

      self.imagens.informEspaciais()
      inf_crs = self.imagens.crs
      inf_transf = self.imagens.transform

      imagem = self.array_probab

      nome = 'probab'

      caminho_down =
self.tabela_propriedades.loc['caminho_download_imagens']['info
rmacao']

      caminho_nome = caminho_down + '/' + nome + '.tif'

      np_imagens = imagem[newaxis,:,:]

      altura = np_imagens.shape[1]
      largura = np_imagens.shape[2]

      quant_bandas = np_imagens.shape[0]
      tipo_dado = np_imagens.dtype
```

```
      with rasterio.open(caminho_nome, 'w',
                         driver = 'GTiff',
                         height = altura,
                         width = largura,
                         count = quant_bandas,
                         dtype = tipo_dado,
                         crs = inf_crs,
                         transform = inf_transf) as down_img:

                         down_img.write(np_imagens)
class Visualizar:

  #Função de inicialização
  def __init__(self, caminho):
    self.imagens = Imagens(caminho)
    self.esc_treino_total = self.imagens.esc_treino_total
    self.caminho_propriedades =
caminho
                          #O caminho para o arquivo de
propriedades
    self.tabela_propriedades =
Propriedades(caminho).tabela_propriedades
                           #A tabela de propriedades
    self.caminho_array = self.imagens.caminho_escolhido
    self.nparrays = [os.path.join(self.caminho_array,x) for x
in os.listdir(self.caminho_array) if '.npy' in x]
    self.empilharArrays()


  #Função para empilhar os np.arrays
  def empilharArrays(self):

    for array in self.nparrays:
      self.np_stack = np.load(array)

    return
```

```
  #Visualização das imagens
  def verImagensGrupo(self):

    self.imagens.informEspaciais()
    self.imagens.prepararImagens()

    lista_nomes = self.imagens.tabela_imagens.index

    # set some plotting parameters
    mpl.rcParams.update({"axes.grid":True,
"grid.color":"gray",
                         "grid.linestyle":'--
','figure.figsize':(10,10),
                         'axes.titlesize':'large'})

    #Editando como vai ser a plotagem das imagens
    linhas = (self.np_stack.shape[0]//3)
    figura, eixos = plt.subplots(nrows=4, ncols=3,
figsize=(50,35))
    figura.tight_layout()

    for indice_imagem, indice_eixo in
enumerate(eixos.flatten()):
      if indice_imagem < self.np_stack.shape[0]:

        indice_eixo.imshow(self.np_stack[indice_imagem],
                           extent=self.imagens.limites)

        indice_eixo.set(title=lista_nomes[indice_imagem])
        barra_escala = ScaleBar(0.125, 'km', dimension = 'si-
length',
                                length_fraction=0.1,location='
lower left')
        indice_eixo.add_artist(barra_escala)

      else:
```

```
        indice_eixo.axis('off') #Não plota a grade após a
última imagem ser plotada

    return plt.show()


  #Função para visualiza uma única imagem
  def verImagemUnico(self):

    mpl.rcParams.update({"axes.grid":True,
"grid.color":"gray",
                         "grid.linestyle":'--
','figure.figsize':(20,20),
                         'axes.titlesize':'large'})
    fig, ax = plt.subplots()

    self.imagens.informEspaciais()
    self.imagens.prepararImagens()
    self.imagens.atualizarTabelaValMinMax()

    self.indice_imagem = int(input('Coloque o valor: '))

    self.imagem = self.np_stack[self.indice_imagem]

    nome =
self.imagens.tabela_imagens.index.tolist()[self.indice_imagem]

    ax.imshow(self.imagem,
              extent=self.imagens.limites)

    vmin=self.imagens.tabela_imagens['valor_min'].tolist()[sel
f.indice_imagem]
    vmax=self.imagens.tabela_imagens['valor_max'].tolist()[sel
f.indice_imagem]

    sm =
plt.cm.ScalarMappable(norm=plt.Normalize(vmin=vmin,vmax=vmax))
```

```
    cbar = fig.colorbar(sm, shrink = 0.40,
spacing='proportional')

    barra_escala = ScaleBar(0.125, 'km', dimension = 'si-
length',
                            length_fraction=0.1,location='lowe
r left')
    ax.add_artist(barra_escala)
    ax.set(title = nome)
    plt.show()


  #Função para baixar uma única imagem
  def baixarImagem(self):

    self.verImagemUnico()

    imagem = self.np_stack[self.indice_imagem]

    nome =
self.imagens.tabela_imagens.index.tolist()[self.indice_imagem]

    caminho_down =
self.tabela_propriedades.loc['caminho_download_imagens']['info
rmacao']

    caminho_nome = caminho_down + '/' + nome + '.tif'

    np_imagens = imagem[newaxis,:,:]

    altura = np_imagens.shape[1]
    largura = np_imagens.shape[2]

    quant_bandas = np_imagens.shape[0]
    tipo_dado = np_imagens.dtype
    sistem_coord = self.imagens.crs
```

```
    transform = self.imagens.transform

    with rasterio.open(caminho_nome, 'w',
                       driver = 'GTiff',
                       height = altura,
                       width = largura,
                       count = quant_bandas,
                       dtype = tipo_dado,
                       crs = sistem_coord,
                       transform = transform) as down_img:

                       down_img.write(np_imagens)
```