POLI-USP labgeo

Laboratórios didáticos de Geoprocessamento

Versão v0.1

Guilherme Fernandes Alves

05 mai., 2024

Índice:

Este repositório contém os laboratórios didáticos da disciplina PTR3311 - Geomática 2 da Escola Politécnica da USP. Os laboratórios são escritos em Python e utilizam o Jupyter Notebook como ambiente de desenvolvimento.

Os notebooks estão disponíveis em formato .ipynb no nosso repositório do GitHub e esta página contém a documentação completa dos laboratórios.

CAPÍTULO 1

Documentação técnica

Aqui você encontra uma documentação mais extensa do que a encontrada dentro dos notebooks. A ideia é que você possa consultar esses documentos para entender melhor os conceitos e as técnicas utilizadas durante os laboratórios.

1.1 Conceitos básicos

Antes de iniciar com os laboratórios, pode ser importante revisar alguns conceitos sobre a disciplina e sobre a linguagem *Python*.

1.1.1 Sobre a disciplina

A disciplina de Geomática 2 (PTR3311) é oferecida pelo Departamento de Engenharia de Transportes (PTR) da Escola Politécnica da USP. Parte dessa disciplina é dedicada ao estudo de técnicas de (geo)processamento de imagens de sensoriamento remoto, e é nesse contexto que este material foi desenvolvido.

O objetivo é apresentar os conceitos básicos de sensoriamento remoto e geoprocessamento, e introduzir o uso de ferramentas computacionais para o processamento de imagens de satélites.

Sensoriamento Remoto

Sensoriamento remoto é a ciência e arte de obter informações sobre um objeto, área, ou fenômeno a partir de dados adquiridos por um dispositivo que não está em contato direto com o objeto, área ou fenômeno investigado [LILLESAND2015].

É também uma tecnologia que utiliza sensores e equipamentos de processamento e transmissão de dados a bordo de aeronaves ou outras plataformas. O objetivo é estudar fenômenos e processos que ocorrem na superfície da Terra, a partir do registro da radiação eletromagnética (REM), adaptado de [NOVO2008].

Geoprocessamento

Geoprocessamento é a disciplina que utiliza técnicas computacionais para a coleta, processamento, análise e visualização de dados geográficos. Abrange uma série de ferramentas e técnicas para o tratamento da informação espacial, sendo fundamental em áreas como planejamento urbano, gestão de recursos naturais, cartografia, e estudos ambientais.

1.1.2 Sobre Python

Python é uma linguagem de programação de alto nível e interpretada, isso significa que não é necessário compilar o código para executá-lo, o interpretador faz isso para você. É uma linguagem de propósito geral, ou seja, pode ser usada para desenvolver qualquer tipo de aplicação, desde um simples *script* até um sistema complexo. Essas características fazem com que Python seja uma linguagem bastante acessível para iniciantes, mas também bastante útil para profissionais experientes.

Os laboratórios deste curso foram escritos para `Python 3.8` ou superior. É recomendável revisar alguns conceitos básicos de programação antes de iniciar. Se você já tem experiência com Python ou outra linguagem de programação, sinta-se à vontade para pular esta parte.

Estrutura de dados

- **Variáveis:**
  - Espaços de memória para armazenar dados que podem ser alterados durante a execução do programa.
  - Em Python, não é necessário declarar o tipo de uma variável antes de utilizá-la.
  - O tipo de uma variável é definido no momento em que um valor é atribuído a ela.
  - Isso aqui é uma variável em Python: `variavel = 1`.
  - Isso aqui é uma variável em Python: `variavel = "eu sou uma string"`.
- **Números:**
  - Incluem inteiros (`int`), números de ponto flutuante (`float`) e números complexos (`complex`).
  - Em Python, são permitidas operações aritméticas entre diferentes tipos de números. Isso é uma flexibilidade que não necessariamente está presente em outras linguagens de programação.
  - Isso aqui é um número inteiro em Python: `numero = 1`.
- **Strings:**
  - Sequências de caracteres delimitadas por aspas simples ou duplas.
  - Em Python, strings são imutáveis, ou seja, não podem ser alteradas após a sua criação.
  - Podem ser concatenadas utilizando o operador +.
  - Podem ser acessadas utilizando índices ou fatiamento.
  - Isso aqui é uma string em Python: `string = "eu sou uma string"`.
- **Listas:**
  - Coleções ordenadas e mutáveis que podem conter itens de diferentes tipos.
  - Em Python, são identificadas por colchetes `[]`.
  - Isso aqui é uma lista em Python: `lista = [1, 2, 3, 4, 5]`.
- **Tuplas:**

  - Coleções ordenadas e imutáveis, utilizadas para agrupar dados relacionados.
  - Em Python, são identificadas por parênteses `()`.
  - Isso aqui é uma tupla em Python: `tupla = (1, 2, 3, 4, 5)`.
- **Dicionários:**
  - Estruturas de dados que armazenam pares chave-valor, permitindo a rápida recuperação de dados.
  - Em Python, são identificados por chaves `{}`.
  - Isso aqui é um dicionário em Python: `dicionario = {"chave1": "valor1", "chave2": "valor2", ...}`.
- **Funções:**
  - Blocos de código que executam uma tarefa específica.
  - Em Python, são definidas utilizando a palavra-chave `def`.
  - Isso aqui é uma função em Python: `def funcao():`.

Orientação a objetos

Orientação a objetos é um paradigma de programação que utiliza `objetos` para representar dados e métodos para interagir com esses dados.

- **Classes:**
  - Estruturas que definem o comportamento e as características dos objetos.
  - Em Python, são definidas utilizando a palavra-chave `class`.
  - Isso aqui é uma classe em Python: `class NomeDaClasse`.
  - No nosso curso, dificilmente vamos criar classes, mas é importante entender o conceito.
- **Objetos:**
  - Instâncias de classes que encapsulam dados e comportamentos.
  - Em Python, são criados utilizando a função `__init__`.
  - Isso aqui é um objeto em Python: `objeto = NomeDaClasse()`.
- **Métodos:**
  - Funções definidas dentro de uma classe que descrevem as ações dos objetos.
  - Em Python, são definidos utilizando a palavra-chave `def`.
  - Um método é uma função, mas nem toda função é um método.

Controle de fluxo

- **Condicionais:**
  - Estruturas que permitem a execução de um bloco de código caso uma condição seja verdadeira.
  - Em Python, são definidas utilizando as palavras-chave `if`, `elif` e `else`.
  - Isso aqui é um condicional em Python: `if condicao:`.
- **Loops:**

  - Estruturas que permitem a repetição de um bloco de código.
  - Em Python, são definidos utilizando as palavras-chave `for` e `while`.
  - Isso aqui é um loop em Python: `for item in lista:`.
  - A estrutura `for` é geralmente mais utilizada e permite iterar sobre uma sequência de itens, como visto no exemplo acima.

- **Funções e métodos nativos:**
  - Funções e métodos que já vêm instalados com o Python e permitem a execução de tarefas específicas.
  - Em geral, são mais eficientes do que escrever o código do zero.
  - São exemplos de funções e métodos nativos: `sum()`, `map()`, `reduce()`, entre outros

Bibliotecas

Bibliotecas são conjuntos de códigos que podem ser reutilizados em diferentes programas, simplificando o desenvolvimento ao fornecer funcionalidades pré-definidas.

Além das bibliotecas nativas do Python, existem bibliotecas de terceiros. No nosso curso, utilizaremos algumas como o `geemap`, `geopandas`, `matplotlib`, que são importantes para trabalhar com geoprocessamento e visualização de dados.

Jupyter Notebook

Jupyter Notebook é uma ferramenta interativa que permite a escrita de código, visualização de resultados e documentação em um único lugar. É amplamente utilizado para análise de dados, aprendizado de máquina e visualização.

Onde pedir ajuda?

Utilize o comando `help()` do Python para obter informações sobre o uso de funções e módulos. Abaixo damos um exemplo de como utilizar o comando `help()` para obter informações sobre a função `print()`.

```
help(print)
```

Participe do fórum de discussão do curso para colaborar e aprender com outros alunos. Em geral, você encontrará respostas para dúvidas frequentes e poderá compartilhar suas experiências com outros alunos. Acesse aqui: https://github.com/Gui-FernandesBR/PTR3311-Python/discussions

O Stack Overflow é um recurso valioso para solucionar dúvidas específicas de programação e encontrar soluções para problemas comuns.

1.2 Laboratório 1

Documentação teórica do primeiro laboratório. Ao final de cada página é possível encontrar referências bibliográficas (quando disponíveis) e também um link para o próximo tópico.

1.2.1 Sobre este repositório

Antes de começar, vale a pena explicarmos o objetivo deste repositório e como utilizá-lo.

Você pode saltar essa leitura e ir direto para os laboratórios caso queira.

Motivação

O Google Earth Engine (GEE) é uma ferramenta poderosa para o processamento de imagens de satélites, mas sua curva de aprendizado pode ser íngreme, especialmente para quem não tem experiência com programação. Este repositório foi criado para facilitar o aprendizado do GEE, fornecendo exemplos didáticos e exercícios práticos.

Principais considerações

O repositório se baseia nas seguintes premissas fundamentais:

1. **Didática:**
   Fornecemos uma série de laboratórios didáticos para facilitar o entendimento e a utilização prática do GEE.
2. **Acadêmico:**
   Estes materiais são utilizados na disciplina PTR3311 - Geomática 2, ministrada na Escola Politécnica da USP para diversos cursos de engenharia.
3. **Colaborativo e Inclusivo:**
   O repositório é de código aberto, e contribuições externas são bem-vindas, seja você um aluno, professor ou um especialista em GEE.
4. **Estrutura Progressiva:**
   Os laboratórios seguem uma ordem que desafia o aprendizado do usuário, começando do básico até níveis mais avançados.
5. **Python como Linguagem de Programação:**
   Optamos por utilizar Python por sua simplicidade, especialmente para aqueles novos na programação. Esta escolha representa uma mudança de nossos laboratórios anteriores em JavaScript.
6. **Plataforma de Aprendizado Interativo:**

   Utilizamos Jupyter Notebooks para uma experiência de aprendizado interativa, permitindo a execução de código em blocos isolados. Graças à compatibilidade com o Google Colab, não é necessário instalar ambientes de programação locais.
7. **Acessibilidade Linguística: Todos os laboratórios estão disponíveis em**
   português, removendo barreiras linguísticas para falantes não nativos de inglês.

1.2.2 Introdução

O Sensoriamento Remoto fornece dados brutos que precisam ser processados até se tornarem compreensíveis. Isso inclui correções radiométricas, geométricas e atmosféricas, além da transformação de imagens.

Para lidar com grandes volumes de dados, a Google criou o *Google Earth Engine (GEE)*, uma plataforma de *Cloud Computing* para trabalhar com imagens de satélites.

Os laboratórios didáticos deste repositório apresentam o GEE, cobrindo desde aspectos básicos como criação de funções e cálculos básicos com objetos no GEE, até aplicações mais avançadas como a classificação supervisionada de imagens.

O primeiro laboratório apresenta o GEE e o *Google Colab*, realizando operações simples e permitindo a visualização de imagens.

1.2.3 Cadastro no Google Earth Engine

Aqui descrevemos o processo passo a passo para criar uma conta no Google Earth Engine (GEE).

Pré-requisitos

- Acesso à Internet.
- Uma conta do Google (existente ou nova).

Passos para Usuários Não-Comerciais

1. Acesse a página oficial do GEE em https://earthengine.google.com/ (testado em 25 de setembro de 2023).
2. Escolha a opção `Without Cloud Project`.
3. Clique em `Sign Up` para iniciar o processo de cadastro.
4. Preencha o formulário apresentado. Não selecione a opção de uso comercial para garantir a ativação da conta.
5. Envie o formulário clicando em `SUBMIT`. Você será redirecionado para uma página de confirmação.
6. Aguarde o recebimento de um e-mail confirmando a ativação da sua conta.

Passos para Usuários Comerciais

1. Usuários comerciais devem acessar o serviço do GEE através do Google Cloud.
2. Crie ou associe um projeto existente no Google Cloud para usar com o GEE seguindo as instruções na página de registro do projeto.
3. Os laboratórios desta disciplina foram criados visando o uso não comercial, portanto as instruções desta seção serão curtas.

Opção para Criação de Nova Conta de Email

1. Visite a URL "https://code.earthengine.google.com/".
2. Se for necessário, crie uma nova conta Google clicando em `Create account` e selecionando `For myself`.
3. Complete o processo de criação da conta e responda ao questionário sobre o uso do GEE.
4. Aceite os Termos e Condições para finalizar a criação da conta.

Associação do GEE a uma Conta de Email Existente

1. Insira o seu e-mail Google existente e clique em `Next`.
2. Você será orientado a associar as credenciais do Earth Engine com sua conta de e-mail existente.
3. Siga o link fornecido para completar a associação.

Dicas para um Processo de Cadastro Suave

- Utilize o navegador Google Chrome.
- Certifique-se de estar conectado com apenas uma conta Google ao se inscrever.
- Prefira usar um e-mail de universidade/organização para se registrar.
- Assegure que a configuração de grupos do Google esteja correta, permitindo que gerentes de grupos o adicionem aos seus grupos.

Figuras

... em breve ... . image:: figs/gee_home.png .. :width: 100% .. :align: center .. :alt: Página inicial do Google Earth Engine

Para mais informações, visite a [Guia do Usuário do Earth Engine](https://developers.google.com/earth-engine/guides).

1.2.4 GEE, Big Data e Computação Paralela

O Google Earth Engine (GEE) é uma plataforma de análise geoespacial baseada na nuvem, otimizada para o processamento de grandes conjuntos de dados. Utilizando uma arquitetura avançada de *Big Data* e técnicas de computação paralela, o GEE é capaz de realizar operações complexas em dados geoespaciais em grande escala e alta velocidade.

Big Data

O conceito de *Big Data* é caracterizado pelos três Vs:

- **Volumoso:** Grandes conjuntos de dados.
- **Variado:** Diversidade de tipos de dados.
- **Veloz:** Necessidade de processamento rápido.

O GEE emprega padrões de computação paralela, como «map» e «reduce», para distribuir eficientemente as operações através de múltiplos processadores.

Padrões de Computação Paralela

Map

O padrão «Map» aplica uma operação a cada item em um conjunto de dados, produzindo um novo conjunto de itens correspondentes, conforme ilustrado na Figura 1.

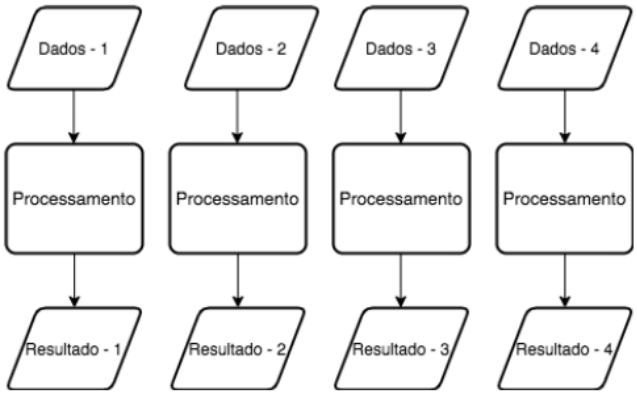

Fig. 1: *Figura 1: Diagrama ilustrativo do padrão Map. Adaptado de* [GRIEBLER2011]

Reduce

O padrão «Reduce», por outro lado, sumariza ou agrega um conjunto de dados em um único valor ou conjunto menor, como mostrado na Figura 2.

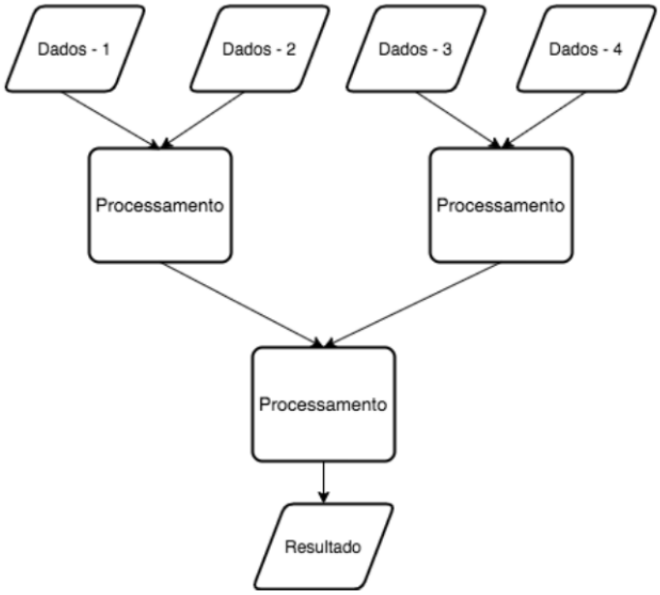

Fig. 2: *Figura 2: Representação esquemática do padrão Reduce. Adaptado de* [GRIEBLER2011]

MapReduce

O modelo «MapReduce» combina os padrões «Map» e «Reduce». O conjunto de saídas gerado pela fase «Map» serve como entrada para a fase «Reduce», permitindo processamento eficiente e escalável em grandes volumes de dados.

1.2.5 Tipos de dados do GEE

As principais classes para armazenamento de dados do GEE são:

- **ee.Image**: Dados raster que armazenam informações numéricas em forma de matriz.
- **ee.ImageCollection**: Coleções de objetos do tipo *Image*.
- **ee.Feature**: Dados vetoriais que contêm geometrias e atributos.
- **ee.FeatureCollection**: Um grupo de objetos do tipo *Feature*.

Dados do tipo Raster

Dados raster são representados como uma matriz de células, cada uma armazenando valores (atributos) e representando *pixels*. Um arquivo raster pode ter mais de um atributo em cada célula, conhecidos como bandas. Sensores de satélite normalmente produzem dados raster com múltiplas bandas, cada uma representando um comprimento de onda do espectro eletromagnético.

Metadados

Metadados são informações sobre o conteúdo de um arquivo raster e são essenciais para sua manipulação. Os metadados incluem detalhes como a inclinação do sol, que pode ser usada para correção radiométrica. Eles também ajudam na busca de dados com características específicas e podem ser encontrados em manuais fornecidos pelos provedores de dados.

1.2.6 Resoluções de imagens de satélite

As imagens de satélite podem apresentar diferentes características que impactam na qualidade da informação. Estas características incluem:

Resolução espacial

A resolução espacial das imagens de satélite é a precisão na captura de detalhes geográficos. Você pode encontrar essa informação na aba «BANDS» das informações de coleções do GEE. Em alguns casos, a resolução é listada como *Resolution* para cada banda. Em outros casos, a resolução é a mesma para todas as bandas, e essa informação fica na parte superior da aba «BANDS».

Resolução espectral

A resolução espectral das imagens é definida pelas faixas de comprimento de onda das bandas e pelo número de bandas na coleção. Você pode verificar isso na coluna *Wavelength* da tabela na aba «BANDS». Note que algumas bandas podem conter informações adicionais além das espectrais.

Resolução temporal

A resolução temporal, que depende do tempo de revisita do satélite, está disponível nas especificações técnicas do satélite, geralmente encontradas em seus manuais. Outra maneira de obter informações sobre a resolução temporal é verificar as datas de coleta das imagens.

Resolução radiométrica

Essa resolução diz respeito à precisão das medições de intensidade de radiação, está documentada nos dados originais do satélite que coletou a coleção. É importante notar que cada coleção pode conter imagens derivadas dos dados originais e, portanto, os valores numéricos podem estar em uma faixa diferente dos valores das imagens originais (*raw data*). Para verificar a resolução radiométrica, é possível examinar o tipo de dado numérico dos pixels das bandas de uma imagem da coleção.

1.2.7 Iniciando o Google Colaboratory (colab)

Sobre o Colab

O Google Colab é uma plataforma gratuita baseada na nuvem para criar, compartilhar e executar notebooks interativos de Python (Jupyter Notebooks). Sua utilização é simples e há recursos explicativos disponíveis online.

Recomendações para o uso do Colab

Aqui estão algumas dicas importantes para o ambiente:

- Use células de código para escrever e executar código Python.
- Comente seu código com «#» para maior legibilidade.
- Utilize atalhos de teclado, como Shift+Enter para execução e Ctrl+; para comentários.
- Busque ajuda em fóruns como o StackOverflow e dos monitores e professores.
- Priorize nomes significativos para as variáveis, funções pequenas e evite duplicação de código.
- Reinicie o ambiente (Runtime > Restart runtime) para problemas persistentes, mas cuidado com a perda de variáveis definidas.
- Mantenha o notebook organizado, limpando a saída das células de código com o botão «Limpar saídas».
- Quando simplesmente digitamos o nome de uma variável, o Colab imprime seu valor na saída da célula de código.

Como rodar nossos laboratórios no Colab

Existem basicamente duas formas de iniciar um notebook deste repositório:

1. Acessando o link do notebook no GitHub e clicando no botão «Open in Colab» no topo da página.
2. Acessando o link do notebook no GitHub e substituindo o prefixo «github.com» por «colab.research.google.com/github».
3. Clonando o repositório localmente e fazendo upload do notebook para o Colab.
4. Fazendo o download do notebook e abrindo-o no ambiente jupyter de sua preferência.

1.2.8 Referencias deste laboratório

Para o laboratório 1, recomenda-se as seguintes referências:

- [GORELICK2017]: Para uma introdução ao Google Earth Engine.
- [GRIEBLER2011]: Sobre padrões de computação paralela.
- [MILOSLAVSKAYA2016]: Sobre Big Data.

1.3 Laboratório 2

Documentação teórica do segundo laboratório.

1.3.1 Introdução

Os fundamentos teóricos básicos para este laboratório são derivados dos apresentados no laboratório anterior, cobrindo o armazenamento de dados em arquivos raster e ferramentas adicionais de estatística e análise exploratória de dados.

Além desses, os conceitos relativos ao ajuste de contraste em composições coloridas com as bandas de imagens podem ser encontrados em [CROSTA1992].

1.3.2 Sentinel-2

O texto a seguir refere-se à coleção de imagens Sentinel-2 disponibilizada pela agência espacial europeia, conhecida pela sigla **ESA**.

Level-1C

Nessa coleção, o termo `Level-1C` determina o tipo do produto disponibilizado pela fornecedora das imagens. O produto `Level-1C` possui em seus pixels o valor reflectância no topo da atmosfera - *Top Of Atmosphere (TOA)* - multiplicado por `10000` (dez mil).

Level-2A

De forma similar, é distribuído o produto `Level-2A`, que apresenta a reflectância da superfície - *Bottom Of Atmosphere (BOA)* - multiplicada por `10000` (dez mil). O `Level-2A` é obtido através da chamada correção atmosférica dos dados do `Level-1C`.

Detalhes sobre o processamento dos dados do Sentinel-2 podem ser obtidos na documentação oficial do produto Level-2A.

Programa COPERNICUS

O programa COPERNICUS é uma iniciativa da União Europeia em parceria com a Agência Espacial Europeia (ESA) para monitoramento ambiental e segurança.

É através do COPERNICUS que a ESA disponibiliza os dados do Sentinel-2.

1.3.3 *Digital Numbers (DN)* e Reflectância

Os valores fornecidos nos produtos do `Sentinel-2` são baseados na reflectância.

Já para as coleções derivadas do `Landsat`, estão disponíveis nas chamadas *Raw*, os valores *Digital Number (DN)*, que são os valores das intensidades, ou nível de cinza, dos pixels das imagens digitais geradas pelo satélite, a partir de onde são calculadas a radiância e as reflectâncias no topo da atmosfera e da superfície.

Intervalo de valores

O intervalo dos valores dos DN's é derivado da quantidade de *bits* utilizada para armazenar um pixel, no caso de uma imagem onde o DN está armazenado em 8 *bits*, os níveis de cinza de uma banda ficam entre 0 e 255 (2^8 - 1).

Essas informações podem impactar no modo como se trabalha com os parâmetros do histograma gerado neste laboratório.

Imagens super saturadas

Cabe aqui uma observação a respeito de um passo do pré-processamento, que deve ser levado em consideração. Caso verifique mais apropriadamente os valores dos pixels, após a reescala, provavelmente encontrará valores fora do intervalo entre 0 e 1. O ideal é se fazer o tratamento desses valores durante o pré-processamento dos dados.

Mascarar pixels super saturados

Uma das maneiras seria simplesmente mascarar os pixels (sendo esses pixels anulados e não utilizados nos cálculos). Para fazer isso, utiliza-se as informações da banda `radsat_qa`, que apresenta, em uma sequência codificada de bits, quais bandas estavam muito saturadas (fenômeno chamado de *oversaturation*) no pixel relativo durante aquela coleta dos dados.

1.4 Laboratório 3

Documentação teórica do terceiro laboratório.

1.4.1 Índices de Vegetação

Os índices de vegetação são ferramentas cruciais no sensoriamento remoto, desempenhando um papel vital na avaliação e monitoramento de coberturas vegetais na Terra.

Estes índices são calculados utilizando-se os dados de reflectância obtidos em diferentes bandas espectrais, principalmente no infravermelho próximo (NIR) e no vermelho (RED).

NDVI

O Normalized difference vegetation index (NDVI), ou Índice de Vegetação por diferença normalizada, é um dos mais populares e amplamente utilizados índices de vegetação.

Fórmula

A fórmula do NDVI é expressa por:

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

onde `NIR` representa a reflectância no infravermelho próximo e `RED` representa a reflectância na banda do vermelho visível.

Esta fórmula foi originalmente proposta por [TUCKER1979].

Exemplo de uso

Abaixo exemplificamos como calcular o NDVI com as imagens do Landsat 9. Vamos utilizar linguagem `python`:

```
# Selecionar as bandas necessárias (valores relativos à imagem Landsat 9)
# Atenção: alterar as bandas de acordo com a imagem utilizada
nir = landsatImage.select('B5') # NIR
red = landsatImage.select('B4') # Vermelho

# Calcular a banda NDVI
ndvi = nir.subtract(red).divide(nir.add(red)).rename('NDVI')

# Adicionar a banda NDVI à imagem do Landsat 9
landsatWithNDVI = landsatImage.addBands(ndvi)

# Exibir a banda NDVI no mapa (Map deve ser um objeto geemap)
Map.addLayer(landsatWithNDVI.select('NDVI'), {'min': -1, 'max': 1}, 'NDVI')
```

Interpretação

O NDVI aproveita o contraste entre as propriedades espectrais da vegetação viva — que normalmente absorve a maior parte da luz visível e reflete grandes quantidades de infravermelho próximo — em comparação com outras superfícies como solo ou construções.

Valores de NDVI mais altos indicam maior densidade da vegetação, enquanto valores mais baixos correspondem a superfícies desprovidas de vegetação saudável, como áreas urbanizadas ou solos expostos [LILLESAND2015].

Além do NDVI, outros índices também são importantes para análises mais específicas:

NDBI

O *Normalized Difference Built-Up Index* (NDBI) é usado para identificar áreas urbanizadas construídas, contrastando as bandas espectrais que são mais refletivas em superfícies artificiais do que em vegetação ou solo.

Fórmula

A fórmula do NDBI é expressa por:

$$NDBI = \frac{(SWIR - NIR)}{(SWIR + NIR)}$$

onde `SWIR` representa a reflectância no infravermelho de onda curta e `NIR` representa a reflectância no infravermelho próximo.

Interpretação

O valor do NDBI varia de -1 a 1, onde valores mais altos indicam maior probabilidade de áreas urbanizadas construídas.

Exemplo de uso

Abaixo exemplificamos como calcular o NDBI com as imagens do Landsat 9. Vamos utilizar linguagem `python`:

```
# Selecionar as bandas necessárias (valores relativos à imagem Landsat 9)
# Atenção: alterar as bandas de acordo com a imagem utilizada
swir = landsatImage.select('B6') # SWIR
nir = landsatImage.select('B5') # NIR

# Calcular a banda NDBI
ndbi = swir.subtract(nir).divide(swir.add(nir)).rename('NDBI')

# Adicionar a banda NDBI à imagem do Landsat 9
landsatWithNDBI = landsatImage.addBands(ndbi)

# Exibir a banda NDBI no mapa (Map deve ser um objeto geemap)
Map.addLayer(landsatWithNDBI.select('NDBI'), {'min': -1, 'max': 1}, 'NDBI')
```

EVI

O *Enhanced Vegetation Index* (EVI) é uma alternativa ao NDVI, otimizado para separar melhor a sinalização da vegetação em áreas de alta densidade foliar, onde o NDVI pode tornar-se saturado. O EVI também corrige algumas distorções atmosféricas e de fundo, proporcionando uma medição mais precisa em áreas com grande quantidade de vegetação [GINCIENE2011].

Ambos os índices, NDBI e EVI, são complementares ao NDVI e oferecem uma perspectiva mais ampla para os estudos ambientais e de planejamento urbano.

Fórmula

A fórmula do EVI é expressa por:

$$EVI = 2.5 * \frac{(NIR - RED)}{(NIR + 6 * RED - 7.5 * BLUE + 1)}$$

onde `NIR` representa a reflectância no infravermelho próximo, `RED` representa a reflectância na banda do vermelho visível e `BLUE` representa a reflectância na banda do azul visível.

Fontes: - USGS Landsat Enhanced Vegetation Index - Enhanced Vegetation Index on Wikipedia

Interpretação

O EVI varia de -1 a 1, onde valores mais altos indicam maior densidade da vegetação.

Exemplo de uso

Abaixo exemplificamos como calcular o EVI com as imagens do Landsat 9. Vamos ver primeiro um exemplo de como fazer isso utilizando `javascript`:

```
// Coeficientes para o cálculo do EVI
var gainFactor = 2.5;
var coefficient1 = 6.0;
var coefficient2 = 7.5;
var L = 1.0;

// Selecionar as bandas necessárias (valores relativos à imagem Landsat 9)
// Atenção: alterar as bandas de acordo com a imagem utilizada
var nir = landsatImage.select('B5'); // NIR
var red = landsatImage.select('B4'); // Vermelho
var blue = landsatImage.select('B2'); // Azul

// Calcular a banda EVI
var evi = nir.subtract(red).multiply(gainFactor)
            .divide(nir.add(red.multiply(coefficient1)).subtract(blue.
↪multiply(coefficient2)).add(L))
            .rename('EVI');

// Adicionar a banda EVI à imagem do Landsat 9
var landsatWithEVI = landsatImage.addBands(evi);
```

(continues on next page)

(continuação da página anterior)

```
// Exibir a banda EVI no mapa
Map.addLayer(landsatWithEVI.select('EVI'), {min: -1, max: 1}, 'EVI');
```

A sintaxe em python é muito semelhante:

```
# Coeficientes para o cálculo do EVI
gainFactor = 2.5
coefficient1 = 6.0
coefficient2 = 7.5
L = 1.0

# Selecionar as bandas necessárias (valores relativos à imagem Landsat 9)
# Atenção: alterar as bandas de acordo com a imagem utilizada
nir = landsatImage.select('B5') # NIR
red = landsatImage.select('B4') # Vermelho
blue = landsatImage.select('B2') # Azul

# Calcular a banda EVI
evi = nir.subtract(red).multiply(gainFactor) \
            .divide(nir.add(red.multiply(coefficient1)).subtract(blue.
↪multiply(coefficient2)).add(L)) \
            .rename('EVI')

# Adicionar a banda EVI à imagem do Landsat 9
landsatWithEVI = landsatImage.addBands(evi)

# Exibir a banda EVI no mapa (Map deve ser um objeto geemap)
Map.addLayer(landsatWithEVI.select('EVI'), {'min': -1, 'max': 1}, 'EVI')
```

1.4.2 Classificação de imagens

O objetivo de uma classificação de imagens é separar os pixels dentro de uma imagem em classes [LILLESAND2015], ou seja, agrupá-los de acordo com algum critério. No caso de imagens de satélite, procura-se padrões espectrais semelhantes de acordo com a reflectância em cada pixel da imagem [LILLESAND2015].

A classificação pode ser feita pixel a pixel individualmente, por regiões ou por objetos [LILLESAND2015]. Para este laboratório trabalharemos com um método pixel a pixel.

Classificação não-supervisionada

Definição

A classificação não-supervisionada não utiliza nenhum tipo de conhecimento prévio como base para classificação, não tendo assim o processo chamado de treinamento [LILLESAND2015].

Algoritmos

Estes tipos de algoritmos checam os pixels e os combinam em grupos baseados em alguma forma na qual eles possam ser comparados.

A questão que fica, nesse caso, é que as classes não possuem uma identidade, apenas suas divisões. Especificamente no caso de imagens de satélite, pode-se pensar em termos de classes espectrais, onde não se sabe o que cada uma representa na superfície e esta associação é feita a posteriori pelo analista.

1.4.3 K-means

Neste laboratório é utilizado um método de agrupamento conhecido como *k-means*.

O que é

O k-means trabalha com base em medidas de similaridade entre os objetos (pixels no caso de uma imagem) a fim de agrupá-los quando são similares, ou próximos entre si. Também existem outros métodos que consideram as medidas de dissimilaridade que ao invés da proximidade, trabalha com a distância (diferença) entre os objetos.

Como os atributos (bandas) possuem valores numéricos e contínuos, se utiliza uma métrica, ou medida, de distância. No caso, o usual, para o k-means com dados numéricos reais é o uso da distância euclidiana no espaço de atributos.

Princípio

O princípio do k-means é buscar uma similaridade alta dentro dos grupos (clusters) e uma similaridade baixa entre os objetos de clusters distintos (BOEHMKE; GREENWELL, 2019).

O algoritmo

De forma resumida, o k-means segue os seguintes passos:

1. Recebe um valor de entrada `k <= n` (onde n é o número de objetos do conjunto de dados), que é o número de grupos a ser formado e os dados que serão tratados;
2. Seleciona de forma aleatória - ou direcionada e otimizada, como no caso do `k-means++` descrito por [ARTHUR2006] - k objetos que serão as centróides iniciais;
3. Calcula a distância dos objetos para as centróides;
4. Cada objeto é associado com a centróide da qual ele tem a menor distância, fazendo assim parte desse grupo;
5. Recalcula as centróides de cada grupo;
6. Verifica se o centróide se deslocou consideravelmente:
7. se não, termina o algoritmo,
8. se sim volta para o passo 3.
9. Um critério de máximo de interações pode ser utilizado para a parada do algoritmo após um certo número de execuções.

WEKA - K-means (GEE)

Esta é uma alternativa oferecida pelo GEE está na classe *ee.Clusterer*. Neste caso, há uma estratégia para a execução da classificação, que diverge do k-means básico.

Seleção de número de clusters ótimo

Ao fim desta seção é apresentado um código, nele pode ser observado um procedimento onde se tenta obter um número ótimo de clusters automaticamente. Para isso, o k-means é processado diversas vezes, para diversos números de clusters a partir de dois e incrementando um a um, para a mesma imagem. Para cada uma das segmentações geradas, é calculada a soma da distância quadrática entre cada pixel e a centróide de seu respectivo cluster. Esse procedimento é feito para todos os clusters gerados por uma clusterização e a soma de todos os valores resultantes de cada cluster é feita. Essa é chamada Soma das Distâncias intra-cluster ou Within Cluster Sum of Squares, conhecido pela sigla WCSS (BOEHMKE ; GREENWELL, 2019). Faz-se isso para as clusterizações criadas de forma iterativa, e depois plota-se um gráfico em que no eixo x estão o número de clusters parametrizados para o k-means e no eixo y o WCSS - Figura 11. Visualmente, o que o gráfico apresenta, é uma distribuição cuja curva indica a partir de quando o esforço computacional deixa de fazer sentido dado o pouco ganho na diminuição da variação dos dados intra-cluster. Esse gráfico pode ser comparado com um outro gráfico onde o que se calcula é a variação entre clusters distintos, em que o objetivo é maximizar as distâncias. Ele também serve de base para o chamado elbow method, ou método do cotovelo (BOEHMKE ; GREENWELL, 2019).

Alternativa para execução do k-means no GEE

Na Figura 18 é apresentado o código para a execução do k-means implementado em "ee.Algorithms". É passado para o método "KMeans()" um dicionário de dados com os parâmetros para o algoritmo. Para a chave "image" é atribuída a imagem que se deseja classificar, em "numClusters", o número de clusters desejado é ajustado. O último parâmetro é particular desta alternativa. Ele apresenta se os números atribuídos como identificação de cada cluster devem ser únicos para toda a imagem (se for true) ou podem ser repetidos em cada pedaço da imagem (false). Esse conceito será entendido melhor no momento de se adicionar a imagem resultante ao mapa. O resultado da execução é uma imagem, onde em cada pixel há o número do cluster ao qual aquele pixel da imagem original pertence.

```
// Executa o k-means implementado em ee.Algorithms na seleção.
var imagemClassificada = ee.Algorithms.Image
                                    .Segmentation
                                    .KMeans(
                                            {
                                              image: imagem,
                                              numClusters:5,
                                              uniqueLabels:false
                                            }
                                    );
```

Para visualizar a imagem classificada, basta utilizar a função `Map.addLayer()` como na Figura 10. O resultado da adição dessa imagem no mapa pode ser visto no exemplo da Figura 19. Na imagem adicionada, pode ser observado claramente limites em que a imagem se divide, gerando uma visualização confusa. Isso ocorre por causa do conceito de tiles do GEE. Uma tile nada mais é do que uma subdivisão da imagem e ela existe para tornar a computação mais eficiente. Essas tiles são carregadas desde as imagens importadas de uma coleção, por isso ao se adicionar uma imagem, algumas vezes, há a impressão de que ela vai sendo gerada aos poucos em quadrados. Cada quadrado desses é uma tile da imagem, que possui dimensões de 256x256 pixels. É possível reduzir as dimensões, porém não aumentar. Essa é uma característica inerente da plataforma do GEE. Se afastar a imagem no mapa com um zoom mais distante, pode ser observado que em certo momento essas tiles não serão mais observadas, sendo a classificação recalculada para a imagem toda (sempre que há um movimento de aproximação e distanciamento do mapa do GEE os mapas adicionados são recalculados). A explicação para isso é a re-escala dos pixels da imagem automaticamente (por exemplo de 30 metros para 60 metros) fazendo 256x256 pixels cobrirem a imagem toda no mapa.

1.4.4 Referências deste laboratório

Para este trabalho, recomenda-se a leitura dos seguintes materiais:

- [ARTHUR2006],
- [BOEHMKE2019],
- [GARETH2013],
- [LILLESAND2015] e
- [TUCKER1979].

1.5 Laboratório 4

Documentação teórica do quarto laboratório.

1.5.1 Introdução

Para este laboratório, recomenda-se a leitura dos seguintes materiais: - [BOEHMKE2019], - [CROSTA1992], - [FRIEDMAN2001], - [GORELICK2017], - [GARETH2013], - [GINCIENE2011], - [LILLESAND2015], - [OLIVEIRA2019].

1.5.2 Remoção de nuvens em imagens

A ideia da remoção de áreas cobertas por nuvens (e sombras) é tornar cada pixel identificado como parte de uma nuvem - ou sombras de nuvens - transparente, para o processamento, enquanto as outras áreas são mantidas na imagem. Esse processo é feito através da *Image Masking*, onde é utilizada uma imagem chamada de máscara.

Máscaras

Uma máscara (*mask*) é uma imagem binária com valores 0 ou 1. A máscara exclui da imagem onde ela é aplicada os pixels em que ela contém o valor 0 (inválidos) e mantém o que contém o valor 1 (válidos).

1.5.3 Mosaicos de imagens

Pré-processamento de imagens são operações que objetivam corrigir imagens degradadas ou distorcidas para criar uma representação mais fidedigna da cena original e melhorar a utilidade da imagem para manipulação posterior. Isso tipicamente envolve o processamento inicial de dados de imagens raw para eliminar o ruído presente nos dados, calibrá-los radiometricamente, corrigir distorções geométricas e expandir ou contrair a extensão de uma imagem através da operação de mosaicking ou subsetting.

Subsetting (subdividir), *layer stacking* (empilhar imagens)

Alguns passos do pré-processamento das imagens que serão utilizadas podem envolver subdividir as imagens para reduzir o volume de dados - o chamado subsetting - o layer stacking - que é utilizado para combinar múltiplas bandas separadas ou layers em uma única imagem, e o mosaicking que é utilizado para juntar diversas imagens que cobrem uma área mais ampla do que apenas uma [LILLESAND2015] (cap 7).

No GEE, os termos composição e mosaico se referem a operações similares. A ideia dessas operações é juntar imagens diferentes em uma só, seja através de operações de agregação de pixels, através de operações matemáticas (composição em imagens sobrepostas na mesma região) ou em imagens que não estão sobrepostas ou que possuem descontinuidades dentro de suas áreas (mosaicos).

Mosaicos e composições

No caso dos mosaicos, as descontinuidades das imagens são preenchidas na ordem em que as imagens estão na coleção conforme pixels não «transparentes» são encontrados na sequência. É como se as imagens estivessem em uma pilha, sendo os vazios da imagem do topo preenchidos pelas imagens abaixo, caso não se preencha pela segunda imagem da pilha, a terceira é utilizada e assim sucessivamente até o total preenchimento de todos os pixels possíveis (alguns podem não ter nenhuma representação na coleção). O termo composição, nesse contexto, não deve ser confundido com a composição entre bandas para visualização no mapa, conforme visto nos laboratórios anteriores.

1.5.4 Classificação supervisionada

Ajustar um modelo de classificação supervisionada pode ser entendido como encontrar um modelo ajustado que relaciona uma resposta às entradas, tentando fazer a predição da resposta de forma mais acurada possível para futuros dados de entrada.

Classificação de imagens de satélite

No caso da classificação supervisionada de imagens de satélite, aqui apresentada, a ideia é associar classes do mundo real aos pixels e posteriormente, dado um pixel sem a associação, encontrar a classe à qual ele pertence.

Random Forest

Árvores de decisão

Árvores de decisão são modelos utilizados quando as variáveis de entrada podem ser tratadas na forma de estruturas de seleção, as chamadas condicionais (se-então), até se chegar no resultado da predição. Essas estruturas de seleção são as regras que são utilizadas para se classificar um objeto.

O que é Random Forest?

O Random Forest nada mais é do que uma forma mais sofisticada derivada das árvores de decisão.

Métodos Preditivos e Bootstrap

No Random Forest diversas árvores são utilizadas para se construir um modelo preditivo mais robusto. Para isso é utilizada a técnica de bootstrap, que é uma ferramenta usada para quantificar a incerteza associada com um determinado método de aprendizagem de máquina (modelo). Ele pode ser usado em situações onde é complicado se calcular o desvio padrão de um conjunto de interesse.

Variação e Consistência com Árvores de Decisão

Uma árvore de decisão pode ter alta variância, o que significa que se o conjunto de treinamento for dividido, consequentemente, cada modelo treinado com uma dessas divisões levará a resultados bem diversos. É neste ponto que o bootstrap entra em cena, para se reduzir a variância de um método de aprendizado de máquina (procedimento chamado de bagging).

Construção e Resultados do Random Forest

Para se obter o resultado desejado, um grupo de árvores é construído usando conjuntos de treinamento a partir da amostragem por bootstrap. Além disso, o Random Forest é capaz de descorrelacionar as árvores geradas. Mais detalhes dessas implementações podem ser encontradas em [FRIEDMAN2001]

1.6 Laboratório 5

Documentação teórica do quinto laboratório.

1.6.1 Avaliação e seleção de modelo

Importância da avaliação do modelo

Para avaliar a performance de um método de classificação, idealmente devem ser utilizados conjuntos de dados separados do que foi utilizado para se treinar o classificador. Isso é importante para uma escolha apropriada do modelo e seus parâmetros, além de se obter uma medida da qualidade do modelo escolhido.

Seleção do modelo

A seleção de modelo tem o objetivo de verificar a performance de diferentes modelos para escolher o melhor. Isso envolve comparar as características de cada modelo, como a eficiência e a precisão na classificação.

Avaliação do modelo

A avaliação de um modelo, por sua vez, é o processo de estimar o erro de predição do modelo escolhido sobre novos dados [FRIEDMAN2001]. Essa etapa é crucial para garantir que o modelo selecionado seja eficaz em cenários reais e com dados que não foram utilizados durante o treinamento.

Divisão da Amostra

Para executar adequadamente as tarefas de seleção e avaliação, uma prática comum é dividir a amostra apresentada em três partes:

1. Conjunto de Treinamento (*Training Dataset*): Geralmente compõe 70% da amostra e é usado para treinar o modelo.
2. Conjunto de Validação (*Validation Dataset*): Representa cerca de 15% da amostra e é utilizado para ajustar os parâmetros do modelo.
3. Conjunto de Teste (*Test Dataset*): Também compreendendo 15% da amostra, é usado para avaliar a performance do modelo final.

Essas proporções não são regras fixas, mas são comumente utilizadas na ciência de dados.

Seleção do melhor modelo

Medida de Qualidade

Para escolher o melhor modelo, é necessário ter uma medida de qualidade para cada um. No contexto da classificação no Google Earth Engine (GEE), a acurácia associada à matriz de confusão é frequentemente usada como essa medido.

Escolhendo o Modelo com Maior Acurácia

Em geral, o modelo que apresenta a maior acurácia é considerado o melhor. Este critério leva em conta a capacidade do modelo de classificar corretamente novos dados, o que é um indicador chave de seu desempenho.

1.6.2 Descrição dos conjuntos

Abaixo é fornecidas uma descrição dos conjuntos de dados utilizados no processo de aprendizado supervisionado.

Conjunto de Treinamento

O conjunto de treinamento é usado para ajustar o modelo (*fit*) a partir de suas características - os pixels com os valores nas bandas e as respectivas classes atribuídas a eles. De forma resumida, pode ser dito que o modelo aprende a partir deste conjunto.

Conjunto de Validação

O conjunto de validação é uma parte da amostra que é utilizada para fornecer uma avaliação sem viés da forma em que um modelo foi ajustado com os dados do conjunto de treinamento e também pode ser utilizado para aprimorar os hiperparâmetros do modelo treinado.

Avaliação e ajuste com o Conjunto de Validação

O conjunto de validação é usado para avaliar um determinado modelo específico.

O modelo validado, embora use esses dados, não aprende com eles. A ideia é usar os resultados obtidos com o conjunto de validação para se voltar e ajustar os hiperparâmetros do modelo, fazendo com que ele eventualmente incorpore indiretamente características do conjunto de validação.

Seleção de Modelo

Outra função do conjunto de validação, que será vista mais adiante, é apresentar uma forma de se selecionar qual o melhor dentre um grupo de modelos aplicados em um mesmo conjunto de treinamento.

Conjunto de Teste

O conjunto de teste é usado para fornecer uma avaliação não enviesada do modelo final e seus parâmetros escolhidos a partir dos procedimentos feitos com o conjunto de validação. Este conjunto é utilizado apenas uma vez no fim do processo, quando o modelo foi treinado com o conjunto de treinamento e validado com o de validação.

1.6.3 Processo de avaliação e seleção do modelo

Na Figura 1, é apresentado um fluxograma detalhando os passos para a escolha de um modelo para um dado conjunto de dados.

Primeiramente os modelos são inicializados com os parâmetros desejados. Em seguida, cada um dos modelos é treinado com o conjunto de treinamentos. Na fase seguinte, aplica-se o modelo treinado no conjunto de validação e a partir do resultado deste passo é obtida uma medida de performance. O melhor modelo é então escolhido e aplicado ao conjunto de testes para se obter a real performance do modelo em um conjunto independente.

Como dito anteriormente, na fase da avaliação no conjunto de validação, caso o modelo tenha respondido exageradamente bem no conjunto treinamento e mal na avaliação sobre o conjunto de validação, pode ter ocorrido o chamado *overfitting* (sobreajuste). Nesse caso, pode se voltar à inicialização para reparametrizar o modelo. Também pode ocorrer do modelo já responder mal na avaliação no próprio conjunto de treinamento, o chamado *underfitting* (sub-ajuste), significando que algo não foi bem parametrizado no treinamento, o que as amostras são insuficientes ou desbalanceadas.

Outra opção que pode ser imaginada a partir do fluxograma é a de que cada modelo inicializado (no exemplo: A, B e C) pode ser um mesmo algoritmo, mas com diferentes configurações de seus parâmetros. Também cabe ressaltar que o que é apresentado na Figura 1 a seguir pode ser generalizado para n modelos e parametrizações.

1.6.4 Exemplos

Alguns exemplos são selecionados para demonstrar a classificação supervisionada no Google Earth Engine.

… em breve …

1.6.5 Referências Bibliográficas

As principais referências utilizadas neste laboratório são: - [BOEHMKE2019], - [CROSTA1992], - [FRIEDMAN2001], - [GORELICK2017], - [GARETH2013], - [GINCIENE2011], - [LILLESAND2015], - [OLIVEIRA2019].

1.7 Indo além

Nas páginas a seguir você encontrará alguns exemplos adicionari de como trabalhar com o `geemap`.

1.7.1 Trabalhando com arquivos shapefile

A biblioteca `geemap` permite trabalhar com arquivos shapefile. Neste guia, exploraremos como importar e exportar shapefiles usando o `geemap`.

A principal referência para esses exemplos é o exemplo oficial disponível no site do geemap.

Importando um shapefile

Primeiro, importamos a biblioteca e criamos um objeto mapa.

```
import geemap

# Criação de um objeto Map
Map = geemap.Map()
Map
```

Em seguida, carregamos e adicionamos o shapefile de países ao mapa.

```
# Caminho para o shapefile de países
countries_shp = '../data/countries.shp'

# Conversão do shapefile para um objeto Earth Engine
countries = geemap.shp_to_ee(countries_shp)

# Adicionando o layer de países ao mapa
Map.addLayer(countries, {}, 'Countries')
```

Agora, fazemos o mesmo para o shapefile dos estados dos EUA.

```
# Caminho para o shapefile dos estados dos EUA
states_shp = '../data/us_states.shp'

# Conversão e adição ao mapa
states = geemap.shp_to_ee(states_shp)
Map.addLayer(states, {}, 'US States')
```

Exportando um shapefile

Por fim, demonstramos como exportar um shapefile a partir de um objeto Earth Engine.

```
# Exportando o layer de estados dos EUA para um shapefile
geemap.ee_to_shp(states, filename='../data/us_states_new.shp')
```

1.7.2 Exportando imagens GeoTIFF

O Google Earth Engine oferece a capacidade de exportar imagens georreferenciadas diretamente para o Google Drive. O formato comum para essas exportações é o GeoTIFF, que é amplamente utilizado para armazenar imagens georreferenciadas.

A seguir, apresentamos os procedimentos para realizar essa operação tanto em JavaScript quanto em Python.

```
import ee

# Autenticação e inicialização do Earth Engine
ee.Authenticate()
ee.Initialize()

# Exporta uma imagem GeoTIFF para o Google Drive
# Substitua 'imagemClassificada' pela imagem desejada
# Substitua 'retanguloSelecao' pelo polígono que define a região da imagem
# Ajuste 'escala' conforme necessário, considerando o tamanho do pixel
# Uma tarefa será iniciada. Acompanhe-a na aba "Tasks".
task = ee.batch.Export.image.toDrive(
    image=imagemClassificada,
    description='classificacao',
    scale=escala,
    region=retanguloSelecao,
    fileFormat='GeoTIFF',
    formatOptions={
        'cloudOptimized': True
    }
)
task.start()
```

CAPÍTULO 2

Laboratórios em Python

Esta é uma coleção de laboratórios de programação em *Python*, desenvolvidos para a disciplina de Geomática 2 da Escola Politécnica da Universidade de São Paulo (USP).

2.1 Laboratório 1

Introdução ao processamento digital de imagens, resoluções, metadados e busca por imagens no GEE

Objetivos:

1. Familiarizar-se com o Google Earth Engine através do Google Colab
2. Manipulação de Dados Geoespaciais provindos de sensores remotos (satélites)
3. Visualização desses Dados Geoespaciais

Ferramentas:

- Python
- Google Earth Engine (GEE)
- Google Colaboratory (Colab)
- geemap

```
[ ]: # OBS: Em Python, usamos o snake_case para nomes de variáveis, seguindo a convenção.
     # Porém, em alguns métodos nativos do GEE voc^e pode encontrar o PascalCase ou o␣
      ↪camelCase.
     # Não se assuste, é normal. Apenas siga o padrão do método que voc^e está usando.
```

2.1.1 Instalação

Precisamos importar as bibliotecas necessárias para trabalharmos nesta sessão do Colab.

[ ]:
```
import ee
import geemap.geemap as geemap
```

Após importar, o earth-engine-api exige que façamos a autenticação com uma conta Google. Para isso, basta executar o código abaixo e seguir as instruções.

[ ]:
```
# Ref: https://developers.google.com/earth-engine/apidocs/ee-authenticate
# Para inicializar a sessão para execução insira o id do projeto em ee.Initialize().
ee.Authenticate()
ee.Initialize(project='id_projeto')
```

A cada vez que você autenticar, um token diferente será gerado. Portanto, não se preocupe em guardar o token gerado.

Não conseguiu instalar?

Felizmente, o Colab já vem com várias bibliotecas instaladas, mas caso exista algum problema, podemos instalar as bibliotecas necessárias através do comando `pip install`, por exemplo:

[ ]:
```
# %pip install geemap
```

Para checar se deu tudo certo agora, você pode rodar:

[ ]:
```
# import sys

# print(
#     f"You have geemap version {geemap.__version__} running on Python {sys.version} and your operating system is {sys.platform}."
# )
```

Agora, se sua dúvida for com relação ao cadastro e geração de token no GEE, tente seguir os passos da documentação oficial (https://developers.google.com/earth-engine/apidocs/ee-authenticate) ou peça ajuda aos monitores.

2.1.2 Parte 1: Introdução

Selecionando área de estudo

Vamos criar um mapa de exemplo. Escolhemos a Escola Politécnica da USP, em São Paulo, SP.

Ao criar o mapa, tente se familiarizar com os controles de zoom e de posição oferecidos pela interface do geemap.

[ ]:
```
lat, lon = -23.5546721, -46.7318389

# OBS: Aqui a latitude vem primeiro.
my_map = geemap.Map(center=(lat, lon), zoom=14)
my_map
```

Em seguida, vamos criar um ponto com as coordenadas de latitude e longitude.

```
[ ]: # OBS: Aqui a longitude é quem vem primeiro, cuidado para não inverter.
poli_usp_point = ee.Geometry.Point(coords=[lon, lat], proj="EPSG:4326")
poli_usp_point
```

Adicionando uma coleção de imagens

Existem inúmeros coleções de imagens disponíveis no GEE. Todos eles podem ser encontrados a partir da documentação oficial: https://developers.google.com/earth-engine/datasets

Vamos começar trabalhando com a coleção de imagens Landsat 7

```
[ ]: # Ref: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C02_T1
↪#description
alias = "LANDSAT/LE07/C02/T1"

# Load the image collection.
dataset = ee.ImageCollection(alias)

print("Voc^e carregou uma ImageCollection com sucesso: ", dataset.args["id"])
```

Nós acabamos de importar um conjunto de imagens, porém não necessariamente vamos analisar todas elas, pois isso pode ser muito custoso computacionalmente. Vamos então filtrar as imagens para selecionar somente as informações que nos interessam.

```
[ ]: # Filtra por datas
## Qualquer imagem que não tenha sido capturada entre `start` e `opt_end` será removida.
## A data `start` é inclusiva, enquanto `opt_end` é exclusiva.
dataset = dataset.filterDate(start="1999-01-01", opt_end="2002-12-31")
dataset
```

```
[ ]: # Filtra por geometria
## neste caso, vamos utilizar o ponto que criamos anteriormente.
## Qualquer imagem que não contenha o ponto será removida.
dataset = dataset.filterBounds(geometry=poli_usp_point)
dataset
```

```
[ ]: # OBS: Caso queira entender melhor o que cada método faz, voc^e pode usar o help do␣
↪Python. Exemplo:

# help(dataset.filterBounds)
```

```
[ ]: # Seleciona somente as bandas que queremos
bandas = ["B3", "B2", "B1"]
true_color321 = dataset.select(selectors=bandas)

# Define par^ametros de visualização para as imagens
true_color321_vis_params = {
    "min": 0,
    "max": 255,
    "gamma": 1.4,
}
```

[ ]:
```
# Temos várias imagens na coleção, vamos escolher apenas uma delas.
## no caso, vamos selecionar a primeira imagem.
# help(true_color321.first)

image = true_color321.first()
image
```

Agora vamos visualizar a coleção de imagens filtrada no mapa que criamos anteriormente.

[ ]:
```
my_map.addLayer(
    ee_object=image,
    vis_params=true_color321_vis_params,
    name="True Color (321)",
    shown=True,
    opacity=0.6,
)
```

Para visualizar o mapa, podemos simplesmente chamar a variável que contém o mapa.

[ ]:
```
my_map
```

Recortando uma parte de uma imagem

Vamos selecionar uma área de estudos para recortar a imagem. Podemos definir um retângulo como geometria de recorte.

Porém, note que existem diversas outras geometrias disponíveis: https://developers.google.com/earth-engine/apidocs/ee-geometry

[ ]:
```
# primeiro cria um ret^angulo a partir das coordenadas das arestas
bbox = ee.Geometry.BBox(west=-46.81, south=-23.4, east=-46.26, north=-23.8)
```

[ ]:
```
# agora recorta a imagem usando o ret^angulo
clipped_image = image.clip(clip_geometry=bbox)
```

[ ]:
```
my_map.addLayer(
    ee_object=clipped_image,
    vis_params=true_color321_vis_params,
    name="Clipped (321)",
    shown=True,
    opacity=1,
)
```

[ ]:
```
my_map
```

Adicionou a mesma imagem várias vezes e não sabe como retirá-las?

Podemos acessar uma tupla com todas as imagens (layers) adicionadas ao mapa, veja como:

[ ]:
```
# my_map.layers
```

Assim podemos remover uma imagem a partir de seu nome, basta utilizar o método `remove_layer`:

[ ]:
```
# help(my_map.remove_layer)
```

[ ]:
```
# my_map.remove_layer(rm_layer="True Color (321) Clipped")
```

Calculando o valor médio dos pixels de uma banda

Vamos aplicar uma função para calcular a média dos valores de uma banda em uma imagem. Para tanto, vamos utilizar uma estratégia de data reduction através do método `reduceRegion`.

[ ]:
```
help(clipped_image.reduceRegion)
```

O parâmetro `reducer` deve ser um algoritmo que a ser utilizado para calcular o valor desejado, no caso a média

O parâmetro `maxPixels` define o limite máximo de pixels a serem processados. Isso é fundamental para evitar sobrecarregar a capacidade de processamento do ambiente de execução.

[ ]:
```
# Vamos calcular o valor médio do pixel para cada banda dentro da região retangular␣
↪selecionada.
valor_medio_pixel_rgb = clipped_image.reduceRegion(
    reducer=ee.Reducer.mean(), maxPixels=1e9
)

print(valor_medio_pixel_rgb.getInfo())
```

2.1.3 Parte 2: Visualização de propriedades das imagens

Certo, você superou a primeira parte do laboratório, agora vamos ver como visualizar as propriedades de imagens.

Para começar, vamos importar uma outra coleção de imagens, ainda do LANDSAT 7, mas agora de um layer diferente

[ ]:
```
# Ref: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C02_T1_L2
landsat_le07_c02_t1_l2 = ee.ImageCollection("LANDSAT/LE07/C02/T1_L2")
landsat_le07_c02_t1_l2
```

Também já vamos recortar a imagem para a área de estudo que definimos anteriormente.

[ ]:
```
landsat_le07_c02_t1_l2 = landsat_le07_c02_t1_l2.filterBounds(geometry=bbox)
landsat_le07_c02_t1_l2
```

Vamos gerar e visualizar um dicionário com as propriedades das imagens da coleção.

[ ]:
```
# Gerar um dicionário de metadados da coleção e armazená-lo em uma variável
dict_landsat = landsat_le07_c02_t1_l2.toDictionary()
dict_landsat
```

Vamos imprimir o nome de todas as propriedades do dicionário, selecionando somente a primeira imagem da coleção.

Vamos imprimir os metadados da primeira imagem da coleção.

[ ]:
```
landsat_le07_c02_t1_l2.first().toDictionary()
```

Vale notar que, além dos metadados do ImageCollection, também temos os metadados de cada imagem individualmente.

Agora a parte mais interessante, vamos calcular o percentual de nuvens da primeira imagem da coleção.

[ ]:
```
# Display the cloud cover percentage of the first image in the collection.
cloud_cover_percentage = (
    landsat_le07_c02_t1_l2.first().get("CLOUD_COVER").getInfo() / 100
)  # varies from 0 to 1

print(
    f"Cloud cover of the first image in the collection: {cloud_cover_percentage:.2%}",
)
```

Trabalhando com metadados

Seleção de imagens em uma coleção utilizando informações dos Metadados

Agora vamos utilizar os meta dados para selecionar imagens de uma coleção, o que pode ser particularmente interessante quando queremos trabalhar com imagens sem nuvens.

Vamos começar filtrando as imagens que possuem menos de 40% de cobertura de nuvens.

[ ]:
```
# Select only the images with cloud cover less than the desired value.
limit = 40
image_col_no_clouds = landsat_le07_c02_t1_l2.filterMetadata(
    name="CLOUD_COVER", operator="less_than", value=limit
)
```

Podemos fazer uma conta rápida de quantos imagens restaram na coleção.

[ ]:
```
print(
    f"Número total de imagens na coleção com cobertura por nuvens "
    + f"menor que o desejado: {image_col_no_clouds.size().getInfo()} imagens",
)
```

Vamos filtrar para um período de datas específico.

[ ]:
```
# Select images within a specified date range.
image_col_2019 = image_col_no_clouds.filterDate(
    start="2019-01-01", opt_end="2020-01-01"
)
```

Vamos contar quantas imagens restaram na coleção.

[ ]:
```
# Print collection details for the selected date range to the console.
print(f"Coleção de imagens no ano de 2019: {image_col_2019.size().getInfo()} imagens")
```

Caso você prefira, pode converter a coleção de imagens, que é um objeto da classe `ee.ImageCollection`, para uma lista de imagens, sendo cada uma um objeto da classe `ee.Image`.

[ ]:
```
# Transform the collection of images for the selected date range into a list.
lst_image_col_2019 = image_col_2019.toList(image_col_2019.size())
lst_image_col_2019
```

Manipulando datas nas imagens

Uma outra forma de selecionarmos a primeira imagem da coleção é acessando o primeiro elemento da lista de imagens. Vale lembrar que em Python os índices começam em 0, então vamos pegar a image de índice 0.

[ ]:
```
# Extract the first image from the list.
primeira_imagem = ee.Image(lst_image_col_2019.get(0))
primeira_imagem
```

Podemos utilizar o método date() para acessar a data de aquisição da primeira imagem.

[ ]:
```
primeira_imagem.date()
```

Vamos fazer o mesmo para a segunda imagem da coleção.

[ ]:
```
# Extract the second image from the list.
segunda_imagem = ee.Image(lst_image_col_2019.get(1))
segunda_imagem
```

Porém, dessa vez vamos utilizar a chave «DATE_ACQUIRED» para acessar a data de aquisição da imagem diretamente do dicionário de metadados. Isso é possível pois a chave «DATE_ACQUIRED» é uma das propriedades da imagem. Veja:

[ ]:
```
# Using the image property directly, print the acquisition date of the second image to␣
↪the console.
print(
    "Data de aquisição da segunda imagem: ",
    segunda_imagem.get("DATE_ACQUIRED").getInfo(),
)
```

Bandas, projeção cartográfica e resolução espacial

Uma lista com os nomes de todas as bandas de uma imagem pode ser obtida com o uso do método `bandNames()`

[ ]:
```
image_col_2019.first().bandNames()
```

A projeção cartográfica das bandas de uma imagem devem ser retornadas com o uso do método `projection()`

[ ]:
```
projecao_b1 = image_col_2019.first().select("SR_B1").projection()
projecao_b1
```

Podemos consultar a resolução da banda 1 através do seguinte método:

[ ]:
```
print("Scale in meters:", ee.Projection.nominalScale(projecao_b1).getInfo())
```

Custom functions

O Google Colaboratory (na verdade, o jupyter) permite que você crie funções personalizadas em diferentes células e as utilize em outras células do seu notebook.

Abaixo vamos definir duas funções de exemplo, atente-se à documentação das funções.

[ ]:
```
def devolve_layer_com_data(imagem):
    """Adicione uma banda à imagem com o valor numérico da data como constante
    nos pixels. A banda é chamada de 'data_numerica'.

    Parameters
    ----------
    imagem : ee.Image
        Uma imagem do GEE.

    Returns
    -------
    ee.Image
        Uma imagem do GEE com uma nova banda chamada 'data_numerica'.
    """
    return imagem.addBands(
        ee.Image(imagem.get("system:time_start")).rename("data_numerica")
    )


def devolve_data(imagem):
    """Extrai a data da imagem. A data é retornada como uma string.

    Parameters
    ----------
    imagem : ee.Image
        Uma imagem do GEE.

    Returns
    -------
    string
        Uma string representando a data da imagem.
    """
    return ee.Image(imagem).date()
```

Agora podemos aplicar as funções que definimos anteriormente em qualquer imagem da coleção.

Vamos utilizar o método map() para aplicar a função `get_date()` em todas as imagens da coleção.

[ ]:
```
datas = image_col_2019.map(devolve_layer_com_data)

# Coleção de imagens com o valor numérico das datas em uma banda adicionada.
datas
```

O código acima retorna uma ImageCollection, pois o argumento passado para o método map() é também uma ImageCollection. Porém, se aplicarmos o map() sobre uma lista, obteremos uma lista como resultado.

[ ]:
```
lst_datas = lst_image_col_2019.map(devolve_data)
```

(continues on next page)

(continuação da página anterior)

```
# Lista com as datas retiradas das imagens na lista de imagens.
lst_datas
```

2.1.4 Parte 3: Indo além

Extraindo valores com o reduce

Essa função será apresentada no próximo laboratório, mas caso queira já entender seu funcionamento

[ ]:
```
# Extract the maximum date from the period.
maior_data = lst_datas.reduce(ee.Reducer.max())
maior_data
```

O valor da data é um inteiro que representa o número de milissegundos desde a meia-noite de 1º de janeiro de 1970. Podemos converter esse valor para uma data legível utilizando o método `ee.Date()` e seu submétodo `format()`.

[ ]:
```
ee.Date(maior_data).format("YYYY-MM-dd").getInfo()
```

Se antes calculamos a data máxima, podemos facilmente calcular a data mínima também

[ ]:
```
menor_data = lst_datas.reduce(ee.Reducer.min())
ee.Date(menor_data).format("YYYY-MM-dd").getInfo()
```

Também podemo contar o número de imagens na coleção

[ ]:
```
contagem_images = datas.size().getInfo()
contagem_images
```

Operações aritméticas no GEE

Podemos utilizar o método `diference()` para calcular a diferença entre duas datas. Neste caso, vamos calcular a diferença entre a data máxima e a data mínima do conjunto de imagens.

[ ]:
```
# Calculate the total number of days between the first and last acquisitions in the␣
↪period.
days_between = ee.Date(maior_data).difference(ee.Date(menor_data), "day")
days_between
```

O número acima representa o número de dias entre a primeira e a última aquisição de imagem da coleção

Agora vamos calcular o a média de dias entre as aquisições de imagens da coleção. Para tanto, vamos dividir o número de dias entre a primeira e a última aquisição pelo número de imagens na coleção. Utilizaremos o método `divide()` para isso.

[ ]:
```
average_days_between = days_between.divide(contagem_images - 1)
average_days_between

# q: por que subtrair 1?
# a: porque a diferença entre a primeira e a última imagem é igual ao número de imagens␣
↪menos 1.
```

Tipos armazenados nas bandas e resolução radiométrica

O método `bandTypes()` retorna um dicionário que contém o tipo de dado de cada banda de uma imagem.

No nosso caso, vamos selecionar somente a primeira imagem da coleção, por isso o uso do método `first()` antes da chamada de `bandTypes()`.

```
[ ]: image_col_2019.filterBounds(bbox).first().bandTypes()
```

O dicionário impresso no console mostra em `max` o maior número que pode ser registrado em um pixel e, de forma similar, em `min` está o valor mínimo.

Além disso, ao lado do nome da banda, o tipo de dado que cada pixel da imagem armazena é determinado, por exemplo um tipo inteiro de 16 bits.

Através dessas informações, pode-se determinar a resolução radiométrica da banda

Sugestões de exercícios

Com intuito de ir ainda mais além, sugere-se a realização dos seguintes treinos:

- Adicionar outra coleção de imagens do Landsat (Landsat 8)
- Importar coleção de imagens do Sentinel-2

2.1.5 Atividade

Para finalizar, vamos fazer um exercício de fixação. Responda às perguntas estabelecidas abaixo, tome cuidado para não alterar o nome das variáveis daqui pra frente, principalmente a variável `MY_FINAL_RESULT`

```
[ ]: p1 = "1 - Qual o seu nome?"
r1 = str("")  # preencha com uma string, exemplo: str("Meu nome")

p2 = "2 - Qual o seu número USP?"
r2 = int()  # preencha com um número inteiro, ex: int(12345678)

p3 = "3 - Qual é o valor médio de pixel na banda 2 da primeira imagem da coleção Landsat?
↪"
r3 = float()  # preencha com um número real

p4 = f"4 - Quantas imagens restaram na coleção após filtrar aquelas com menos de 40% de␣
↪cobertura de nuvens?"
r4 = int()  # preencha com um número inteiro

p5 = "5 - Qual é a resolução espacial da banda 1 da primeira imagem da coleção landsat_
↪le07_c02_t1_l2?"
r5 = float()  # preencha com um número real

p6 = "6 - Qual é a diferença em dias entre a data de aquisição da primeira e da última␣
↪imagem na coleção de 2019?"
r6 = int()  # preencha com um número inteiro

p7 = "7 - Qual a data de aquisição da 3ª imagem da coleção landsat_le07_c02_t1_l2?"
```

(continues on next page)

(continuação da página anterior)

```
r7 = str("")  # preencha com uma string formato AAAA-MM-DD

p8 = "8- Qual a resolução radiométrica da banda 1 da 1ª imagem da coleção Landsat?"
r8 = float()  # preencha com um número real
```

[ ]:
```
MY_FINAL_RESULT = {
    "p1": r1,
    "p2": r2,
    "p3": r3,
    "p4": r4,
    "p5": r5,
    "p6": r6,
    "p7": r7,
    "p8": r8,
}

MY_FINAL_RESULT
```

Antes de ir, verifique se está tudo certo com a sua resolução:

[ ]:
```
class ValidaResposta:
    """Classe que vai receber o dicionario MY_FINAL_RESULT e validar se as
    respostas t^em o formato correto.
    """

    def __init__(self, dicionario_respostas):
        self.dicionario_respostas = dicionario_respostas
        self.perguntas_types = {
            "p1": str,
            "p2": int,
            "p3": float,
            "p4": int,
            "p5": float,
            "p6": int,
            "p7": str,
            "p8": float,
        }

    def valida_respostas(self):
        """Valida se as respostas estão no formato correto e não estão vazias.

        Returns
        -------
        bool
            True se todas as respostas estão no formato correto e não estão vazias.
            False se alguma resposta não está no formato correto ou está vazia.
        """
        for pergunta, resposta in self.dicionario_respostas.items():
            expected_type = self.perguntas_types.get(pergunta)
            if (
```

(continues on next page)

(continuação da página anterior)

```
                expected_type is None
                or not isinstance(resposta, expected_type)
                or (
                    isinstance(resposta, str)
                    and resposta in ["Meu Nome", "", "Minha resposta"]
                )
            ):
                print(
                    f"A resposta para a {pergunta} não está no formato correto ou está␣
↪vazia."
                )
                return False

        return True


validador = ValidaResposta(MY_FINAL_RESULT)
if validador.valida_respostas():
    print(">>> Todas as respostas estão no formato correto, parabéns.\n")
else:
    raise ValueError("Alguma resposta não está no formato correto ou está vazia.")
```

Quer levar esse notebook com você?

[ ]:
```
# !pip install nbconvert
```

[ ]:
```
!jupyter nbconvert --to html "lab1.ipynb" --template classic
```

2.2 Laboratório 2

Estatísticas, histogramas, contraste e composições de imagens no GEE

Objetivos:

1. Operações para calcular estatísticas dos valores dos pixels nas bandas de um raster e gerar gráficos de suas distribuições
2. Extrair valores de um pixel em uma imagem
3. Ajustar o contraste de uma imagem adicionada ao mapa e como formar composições coloridas com as bandas dos dados

2.2.1 Introdução

```
[ ]: import ee
     import geemap.geemap as geemap
     import matplotlib.pyplot as plt
```

```
[ ]: # Ref: https://developers.google.com/earth-engine/apidocs/ee-authenticate
     # Para inicializar a sessão para execução insira o id do projeto em ee.Initialize().
     ee.Authenticate()
     ee.Initialize(project='id_projeto')
```

```
[ ]: %matplotlib inline
```

2.2.2 Desenvolvimento

Definir um ponto na cidade de São Paulo

```
[ ]: lat, lon = -23.5546721, -46.7318389
     poli_usp_point = ee.Geometry.Point(coords=[lon, lat], proj="EPSG:4326")
```

Definir um bbox na cidade de São Paulo

Observação: Fazer a seleção de um polígono retangular pequeno na área de São Paulo para que o processamento fique mais rápido.

```
[ ]: sao_paulo_box = ee.Geometry.BBox(
         west=-46.81,
         south=-23.5,
         east=-46.654,
         north=-23.7,
     )
```

Preparando a coleção

Importar a coleção de images "Sentinel-2 MSI: MultiSpectral Instrument, Level-1C"

```
[ ]: colecao_sentinel2 = ee.ImageCollection("COPERNICUS/S2_SR")
     # Veja a documentação em: https://developers.google.com/earth-engine/datasets/catalog/
     ↪COPERNICUS_S2_SR#bands
```

Selecionar apenas as imagens que contém o ponto selecionado

```
[ ]: img_histograma = colecao_sentinel2.filterBounds(poli_usp_point)
```

Filtrar apenas imagens em um intervalo de datas desejado

```
[ ]: start_date = ee.Date("2019-01-01")
     end_date = ee.Date("2020-01-01")
     img_histograma = img_histograma.filterDate(start_date, end_date)
```

Filtrar apenas as imagens com menos de 20% de cobertura de nuvens

```
[ ]: img_histograma = img_histograma.filterMetadata(
         "CLOUDY_PIXEL_PERCENTAGE", "less_than", 20
     )
```

Selecionar apenas a primeira imagem da coleção, pois precisamos de somente uma imagem para o restante da prática

```
[ ]: img_histograma = img_histograma.first()
```

Selecionar somente algumas das bandas para análise.

```
[ ]: img_histograma = img_histograma.select(["B2", "B4", "B3"])
```

Fazer um recorte (*clip*) da imagem para a área o retângulo que definimos anteriormente

```
[ ]: img_histograma = img_histograma.clip(sao_paulo_box)
```

Criar um mapa através do `geemap`, biblioteca de visualização de dados geoespaciais

```
[ ]: my_map = geemap.Map(center=[lat, lon], zoom=11)
```

Vamos adicionar uma banda da imagem ao mapa para visualização, isso gerará uma imagem monocromática

```
[ ]: my_map.addLayer(
         ee_object=img_histograma.select("B2"),
         vis_params={},
         name="Recorte São Paulo - Banda B2",
     )
     # O segundo par^ametro da função, que ajusta a visualização, está vazio para manter os␣
     ↪valores do padrão.
     # O terceiro par^ametro é o título para a camada que é criada no mapa pela função.
```

Para visualizar o mapa

```
[ ]: my_map
```

Podemos checar o valor da banda B2 para o ponto que criarmos anteriormente:

```
[ ]: value = img_histograma.reduceRegion(
         reducer=ee.Reducer.mean(), geometry=poli_usp_point, scale=10
     )

     print("Valor de cada banda no ponto selecionado:")
     print(value.getInfo())
```

Estatísticas dos dados de uma imagem no GEE

Calcular o desvio padrão dos valores dos pixels de uma imagem com o `Reducer.stdDev()`

```
[ ]: dp_pixels_bandas = img_histograma.reduceRegion(
         reducer=ee.Reducer.stdDev(), maxPixels=1e9
     )
     print("Desvio Padrão: ", dp_pixels_bandas.getInfo())

     # OBS: O par^ametro maxPixels é necessário para evitar erros de memória.
```

Calcular a média dos valores dos pixels de uma imagem com o `Reducer.mean()`

```
[ ]: media_pixels_bandas = img_histograma.reduceRegion(
         reducer=ee.Reducer.mean(), maxPixels=1e9
     )
     print("Média: ", media_pixels_bandas.getInfo())
```

Calcular a variância dos valores dos pixels de uma imagem com o `Reducer.variance()`

```
[ ]: variancia_pixels_bandas = img_histograma.reduceRegion(
         reducer=ee.Reducer.variance(), maxPixels=1e9
     )
     print("Vari^ancia: ", variancia_pixels_bandas.getInfo())
```

Escalonando os valores da imagem

Os valores das coleções do Sentinel-2 disponíveis no GEE são os valores da reflectância multiplicados por 10.000 (dez mil). Precisamos dividir os valores por 10.000 para obter os valores de reflectância, que devem obrigatoriamente estar entre 0 e 1.

```
[ ]: img_histograma_nao_escalonada = img_histograma.divide(ee.Image.constant(10000))
```

Gerando histogramas

Vamos gerar histogramas para visualizar a distribuição dos valores dos pixels de uma imagem. Porém antes de gerar o histograma, vamos recalcular os valores das estatísticas para a imagem escalonada.

```
[ ]: dp_pixels_bandas = img_histograma_nao_escalonada.reduceRegion(
         reducer=ee.Reducer.stdDev(), maxPixels=1e9
     )
     print("Desvio Padrão Imagem Não Escalonada: ")
     for key, value in dp_pixels_bandas.getInfo().items():
         print(f"\tBanda {key}: {value:.4f}")
```

```
[ ]: media_pixels_bandas = img_histograma_nao_escalonada.reduceRegion(
         reducer=ee.Reducer.mean(), maxPixels=1e9
     )
     print("Média Imagem Não Escalonada: ")
     for key, value in media_pixels_bandas.getInfo().items():
         print(f"\tBanda {key}: {value:.4f}")
```

```
[ ]: variancia_pixels_bandas = img_histograma_nao_escalonada.reduceRegion(
         reducer=ee.Reducer.variance(), maxPixels=1e9
     )
     print("Vari^ancia Imagem Não Escalonada: ")
     for key, value in variancia_pixels_bandas.getInfo().items():
         print(f"\tBanda {key}: {value:.4f}")
```

Agora, para criação do histograma, vamos selecionar somente a banda B2 da imagem

```
[ ]: banda_B2 = img_histograma_nao_escalonada.select("B2")
```

Utilizamos o `ee.Reducer.histogram()` para gerar o histograma.

```
[ ]: # maxBuckets: é o número máximo de buckets (colunas) no histograma. Esse valor deve ser
     ↪uma pot^encia de 2, caso não seja, será arredondado para a pot^encia de 2 mais próxima.
     # minBucketWidth: é a largura mínima de cada bucket (coluna) do histograma.
     # maxPixels: é o número máximo de pixels que serão usados para calcular o histograma.
     histograma_b2 = banda_B2.reduceRegion(
         reducer=ee.Reducer.histogram(maxBuckets=1000, minBucketWidth=0.00001), maxPixels=1e9
     ).getInfo()["B2"]
```

A variável `histograma` é um dicionário, e dela vamos extrair as listas necessárias para realizar o gráfico

```
[ ]: frequencias_b2 = histograma_b2["histogram"]
     bins_b2 = histograma_b2["bucketMeans"]
```

Finalmente, vamos plotar o histograma com o módulo `matplotlib.pyplot` (ou `plt`) que foi importado no início do notebook.

```
[ ]: fig, ax = plt.subplots(figsize=(9, 5))
     ax.bar(bins_b2, frequencias_b2, width=bins_b2[1] - bins_b2[0], color="gray", alpha=0.7)
     ax.plot(bins_b2, frequencias_b2, color="blue", linewidth=2, alpha=1, label="B2")
     ax.set_title("Histograma para reflect^ancia em B2")
     ax.set_xlabel("Reflect^ancia")
     ax.set_ylabel("Frequ^encia")
     ax.set_xlim(0, 1)
     ax.grid(True, alpha=0.3)
     ax.legend()
     plt.show()
```

Gerando histograma para várias bandas

Primeiro vamos obter o histograma para todas as bandas da imagem de uma vez só. Para isso, basta alterar a imagem que estamos utilizando para a imagem original (sem filtrar a banda B2).

```
[ ]: histograma = img_histograma_nao_escalonada.reduceRegion(
         reducer=ee.Reducer.histogram(maxBuckets=2**9, minBucketWidth=0.001), maxPixels=1e9
     ).getInfo()
```

Utilizar um `for` *loop* do Python para gerar diferentes histogramas para cada banda da imagem.

```
[ ]: fig, ax = plt.subplots(figsize=(9, 5))

     for i, banda in enumerate(["B2", "B4", "B3"]):
         frequencias = histograma[banda]["histogram"]
         bins = histograma[banda]["bucketMeans"]

         ax.bar(bins, frequencias, width=bins[1] - bins[0], alpha=0.3)
         ax.plot(bins, frequencias, linewidth=2, label=banda)
         ax.set_title(f"Histograma da reflect^ancia em diferentes bandas")
         ax.set_xlabel("Reflect^ancia")
         ax.set_ylabel("Frequ^encia")
         ax.set_xlim([0, 1])
```

(continues on next page)

(continuação da página anterior)

```
plt.grid(True, alpha=0.3)
plt.legend(loc="upper right")
plt.show()
```

Ajustando o contraste de uma imagem

A imagem que visualizamos anteriormente está com o contraste muito baixo, pois foi adicionada com os valores padrões. Para ajustar, podemos seguir dois caminhos:

1. Editar o parâmetro «Range» no `control_layer` do mapa, que fica no canto superior direito do mapa
2. Remover o *layer* e adicionar novamente com os parâmetros desejados

Vamos seguir o caminho 2 na célula a seguir, porém o resultado é o mesmo.

[ ]:
```
# Remove a camada que tenha o nome "Recorte São Paulo - Banda B2"
# TODO: deve ter uma forma melhor de fazer isso sem usar comprehension list
my_map.remove_layer(
    x for x in my_map.layers if x.name == "Recorte São Paulo - Banda B2"
)

# Define par^ametros de visualização
vis_params = {
    "bands": ["B2"],
    "min": 0,
    "max": 1,
    "gamma": 1,
}

# Adiciona a imagem novamente ao mapa
my_map.addLayer(
    ee_object=img_histograma_nao_escalonada.select("B2"),
    vis_params=vis_params,
    name="Recorte São Paulo - Banda B2",
)
```

Finalmente, para visualizar o mapa com o contraste ajustado, basta executar a célula abaixo.

[ ]:
```
my_map
```

Composição colorida com ajuste de contraste simples

Eu sei que vocês estão pensando: «Mas eu quero ver uma imagem colorida, não uma imagem em tons de cinza!». Para isso vamos utilizar o método `addLayer()` novamente, porém agora vamos adicionar as bandas RGB da imagem.

As bandas RGB são as bandas 4, 3 e 2, respectivamente. Essa informação é retirada da documentação da coleção Sentinel-2 no GEE: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR

[ ]:
```
vis_params = {
    "bands": ["B4", "B3", "B2"],
    "min": 0.04,
    "max": 0.4,
```

(continues on next page)

(continuação da página anterior)

```
    "gamma": 1,
}
# Para escolher os valores de "min" e "max", uma alternativa é utilizar os
# histogramas gerados anteriormente e escolher visualmente os pontos de corte

my_map.addLayer(
    ee_object=img_histograma_nao_escalonada.select(["B4", "B3", "B2"]),
    vis_params=vis_params,
    name="RGB",
)
```

[ ]:
```
my_map
```

2.2.3 Indo além

Composição colorida com ajustes de contraste individuais

É possível fazer um mapeamento mais preciso de máximos e mínimos para cada banda, e assim ajustar de forma mais apropriada o contraste da imagem que será exibida na interface do GEE.

Além disso, vamos utilizar os histogramas que geramos anteriormente para definir os valores de máximos e mínimos para cada banda de forma automática.

Para começar, vamos calcular os valores de máximos para exibição de cada banda. Vamos definir que o valor máximo de cada banda será a média mais uma vez o desvio padrão.

[ ]:
```
max_intervalos = []

media_values = media_pixels_bandas.getInfo().values()
dp_values = dp_pixels_bandas.getInfo().values()

for mean, dp in zip(media_values, dp_values):
    max_intervalos.append(mean + 2.5 * dp)

# Obs.: A função zip() retorna um iterador de tuplas, onde a i-ésima tupla contém
# o i-ésimo elemento de cada um dos argumentos sequenciais ou iteráveis.

max_intervalos
```

Também precisamos definir os valores mínimos para cada banda. Apenas para nos familiarizarmos com a sintaxe, dessa vez vamos evitar utilizar o `for` *loop* do Python, em vez disso, vamos utilizar compreensão de listas (*list comprehension*). Note que o resultado é o mesmo, apesar de ser um pouco mais difícil de ler, mas é uma forma de escrever código mais «pythônica».

[ ]:
```
min_intervalos = [
    mean - 2.5 * dp
    for mean, dp in zip(
        media_pixels_bandas.getInfo().values(), dp_pixels_bandas.getInfo().values()
    )
]
min_intervalos
```

Agora adicionamos a imagem ao mapa com os valores de máximos e mínimos definidos para cada banda.

[ ]:
```
vis_params = {
    "bands": ["B4", "B3", "B2"],
    "min": min_intervalos,
    "max": max_intervalos,
    "gamma": 1.4,
}

my_map.addLayer(
    ee_object=img_histograma_nao_escalonada.select(["B4", "B3", "B2"]),
    vis_params=vis_params,
    name="RGB advanced",
)
```

E finalmente podemos visualizar o mapa com a imagem colorida e com o contraste ajustado.

[ ]:
```
my_map
```

2.2.4 Atividade

Assim como no laboratório anterior, preencha os campos abaixo e submeta este *notebook* para avaliação.

[ ]:
```
p1 = "1 - Qual o seu número USP?"
r1 = int()  # preencha com um número inteiro, ex: int(12345678)

p2 = "2 - Qual foi o valor médio de pixel na banda 2?"
r2 = int()  # preencha com um número inteiro, ex: int(12345678)

p3 = "3 - Calcule o desvio padrão da banda 5."
r3 = float()  # preencha com um número real

p4 = "4 - O que a banda 5 representa no conjunto de dados Sentinel-2? (ver a␣
↪documentação)"
r4 = int()  # preencha com um número inteiro
```

Não altere a célula abaixo, apenas execute-a para carregar o formulário de submissão.

[ ]:
```
MY_FINAL_RESULT = {
    "p1": r1,
    "p2": r2,
    "p3": r3,
    "p4": r4,
}

MY_FINAL_RESULT
```

2.3 Laboratório 3

Cálculo do NDVI e classificação não-supervisionada de imagens no GEE

Objetivos:

1. Cálculo do NDVI no GEE
2. Classificação não supervisionada de imagens com k-means no GEE
3. Cálculo do número ideal de clusters

2.3.1 Introdução

[ ]:
```
import time

import ee
import geemap.geemap as geemap
import matplotlib.pyplot as plt
```

[ ]:
```
# Ref: https://developers.google.com/earth-engine/apidocs/ee-authenticate
# Para inicializar a sessão para execução insira o id do projeto em ee.Initialize().
ee.Authenticate()
ee.Initialize(project='id_projeto')
```

[ ]:
```
%matplotlib inline
```

2.3.2 Desenvolvimento

Importando a coleção

Vamos utilizar imagens que contenham valores de reflectância em seus pixels (ao invés de valores de DN). Precisaremos de uma única imagem para executarmos nossa classificação.

[ ]:
```
lat, lon = -23.5546721, -46.7318389
poli_usp_point = ee.Geometry.Point(lon, lat)
```

[ ]:
```
dataset = (
    ee.ImageCollection("LANDSAT/LC09/C02/T1_L2")
    .filterDate("2010-01-01", "2024-02-01")
    .filterBounds(poli_usp_point)
)
```

Vamos fazer uma pouco usual porém importante, que é aplicar um *scaling factor* na coleção de imagens para

[ ]:
```
# Ref.: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC09_C02_T1_
↪L2#colab-python
# Applies scaling factors.
def apply_scale_factors(image):
    optical_bands = image.select("SR_B.").multiply(0.0000275).add(-0.2)
    thermal_bands = image.select("ST_B.*").multiply(0.00341802).add(149.0)
```

(continues on next page)

(continuação da página anterior)

```
    return image.addBands(optical_bands, None, True).addBands(thermal_bands, None, True)


dataset = dataset.map(apply_scale_factors)
```

Ordena as imagens de acordo com a cobertura de nuvens, depois pega a primeira imagem da lista e a armazena na variável `image`. Esta deve ser a imagem com menor cobertura de nuvens.

```
[ ]: dataset = dataset.sort("CLOUD_COVER", True)
imagem = dataset.first()
```

Apenas para confirmação, vamos plotar o histograma das bandas principais da imagem e verificar que realmente estamos trabalhando com valores de reflectância entre 0 e 1.

```
[ ]: histograma = imagem.reduceRegion(
    reducer=ee.Reducer.histogram(maxBuckets=2**9, minBucketWidth=0.001), maxPixels=1e9
).getInfo()

fig, ax = plt.subplots(figsize=(9, 5))

for i, banda in enumerate(["SR_B4", "SR_B3", "SR_B2"]):
    frequencias = histograma[banda]["histogram"]
    bins = histograma[banda]["bucketMeans"]

    ax.bar(bins, frequencias, width=bins[1] - bins[0], alpha=0.3)
    ax.plot(bins, frequencias, linewidth=2, label=banda)
    ax.set_title(f"Histograma da reflect^ancia em diferentes bandas")
    ax.set_xlabel("Reflect^ancia")
    ax.set_ylabel("Frequ^encia")
    ax.set_xlim([0, 1])

plt.grid(True, alpha=0.3)
plt.legend(loc="upper right")
plt.show()
```

Utiliza as 3 bandas principais (vermelho, verde e azul) para visualizar a imagem e visualiza o mapa

```
[ ]: imagem_vis = {
    "min": 0.0,
    "max": 0.3,
}

my_map = geemap.Map(center=[lat, lon], zoom=10)

my_map.add_layer(imagem.select(["SR_B4", "SR_B3", "SR_B2"]), {}, "Landsat9")

my_map
```

Dessa vez seremos mais ousados... Você deverá selecionar um polígono no mapa do `geemap` e extrair as coordenadas do polígono para utilizarmos como região de interesse (ROI) para a extração da imagem.

Escolha uma região que seja coberta predominantemente por vegetação. Regiões muito grandes podem deixar o processamento bastante lento, então tente escolher uma região de tamanho moderado.

[ ]:
```
polygon = my_map.draw_last_feature
polygon.getInfo()
```

Corta a imagem para apenas o polígono selecionado e visualiza o mapa

[ ]:
```
clipped = imagem.clip(polygon)
my_map.addLayer(clipped, imagem_vis, "Landsat9")
my_map
```

Cálculo do NDVI

Vamos realizar algumas operações aritméticas para calcular o NDVI da imagem.

Primeiramente deve-se extrair as bandas do NIR e do vermelho e colocá-las em variáveis separadas.

[ ]:
```
# Veja na documentação o significado de cada banda
imagem_banda_nir = clipped.select("SR_B5")  # Near Infrared (NIR)
imagem_banda_vermelho = clipped.select("SR_B4")  # Red
```

Adicionar uma banda NDVI à imagem. A fórmula do NDVI primeiro faz a diferença entre os valores da banda NIR e da banda do vermelho visível, depois o valor resultante é dividido pela soma das mesmas duas bandas.

[ ]:
```
clipped = clipped.addBands(
    imagem_banda_nir.subtract(imagem_banda_vermelho)
    .divide(imagem_banda_nir.add(imagem_banda_vermelho))
    .rename("NDVI"),  # NDVI será o nome da nova banda
    ["NDVI"],
)
```

Podemos conferir que no objeto `clipped` foi adicionada uma banda NDVI.

[ ]:
```
clipped
```

Adicionar o NDVI como um *layer* no mapa

[ ]:
```
my_map.addLayer(clipped, {"bands": ["NDVI"], "min": -1, "max": 1}, "NDVI")
```

Visualizar o mapa novamente

[ ]:
```
my_map
```

Execução do k-means

Há duas maneiras de se executar o k-means junto do GEE:

1. Uma delas está dentro de `ee.Algorithms`.
2. A outra é baseada na implementação do pacote WEKA dentro dos algoritmos da classe `ee.Clusterer`.

Vamos trabalhar com a segunda opção por enquanto, e depois podemos experimentar a primeira.

[ ]:
```
escala = 30  # 30 metros, ver descrição da coleção de imagens
bandas = ee.List(["SR_B2", "SR_B3", "SR_B4", "SR_B5", "SR_B6", "SR_B7"])
```

(continues on next page)

(continuação da página anterior)

```
roi = clipped.select(bandas)  # Region of interest
roi
```

Primeiramente, vamos selecionar uma amostra de pixels com o método `sample()` chamado a partir da imagem que será classificada, que neste caso será a imagem `clipped`

[ ]:
```
# Selecionar uma amostra para executar o k-means
amostra_treinamento = roi.sample(
    region=roi.geometry(),  # redundante fazer isso, mas é para ensinar que é possível␣
↪selecionar uma região menor
    scale=escala,
    numPixels=5000,
)
```

O resultado deve ser uma FeatureCollection com `numPixels` amostras.

[ ]:
```
amostra_treinamento
# TODO: acho que seria possível criar um pd.Dataframe com os dados da amostra abaixo,
# Isso facilitaria bastante a fazer o k-means lá na frente, mas o desafio maior é como␣
↪voltar para o ee.ImageCollection depois
```

Executando a clusterização de k-means

[ ]:
```
k = 5  # número de clusters (altere se quiser testar diferentes valores)
clusters_imagem = ee.Clusterer.wekaKMeans(k).train(amostra_treinamento)
```

[ ]:
```
clusters_imagem
```

Agora classificamos os pixels de acordo com o resultado do k-means

[ ]:
```
# Classificando os pixels de acordo com o resultado do k-means da amostra
imagem_classificada_weka = roi.cluster(clusters_imagem)
```

[ ]:
```
imagem_classificada_weka
```

Vamos adicionar a nova imagem ao mapa para visualizarmos o resultado da classificação

[ ]:
```
# Adicionando a imagem classificada no mapa do GEE
# Vamos colorir aleatoriamente, mas voc^e pode alterar as cores depois
my_map.addLayer(imagem_classificada_weka.randomVisualizer(), {}, "k-means")
```

Visualizar o mapa novamente

[ ]:
```
my_map
```

2.3.3 Indo além

Seleção de número de clusters ótimo

Vamos tentar encontrar, de forma programática, o número de clusters ótimo para a classificação. Para tanto, precisamos rodar o k-means para diferentes valores de k e calcular o erro quadrático médio (MSE) para cada um deles.

Antes de mais nada, vamos criar uma função que realiza o k-means dado um número de clusters, a amostra de treinamento e uma imagem.

[ ]:
```
def executa_kmeans(k: int, amostra: ee.FeatureCollection, img: ee.Image):
    """Executa o k-means na imagem 'img' com 'k' clusters, usando 'amostra'
    como amostra de treinamento."""
    # Initialization method to use.0 = random, 1 = k-means++, 2 = canopy, 3 = farthest␣
↪first.
    return img.cluster(
        ee.Clusterer.wekaKMeans(nClusters=k, init=1, fast=True).train(amostra)
    )
```

Agora vamos para a parte mais legal: uma função para calcular a soma dos quadrados das distâncias dos pixels para o centroide do cluster ao qual ele pertence. Essa função é chamada de soma dos quadrados dentro do cluster (within cluster sum of squares, ou WCSS).

[ ]:
```
def wcss(img_kmeans):
    """
    Esta função calcula a soma dos quadrados intra-clusters (Within Cluster Sum of␣
↪Squares)
    para cada imagem clusterizada. Créditos: Leonardo Godoy
    """
    img_kmeans = ee.Image(img_kmeans)

    # Encontra o ID de cluster máximo na imagem
    max_id_cluster = img_kmeans.reduceRegion(
        reducer=ee.Reducer.max(), maxPixels=1e12, bestEffort=False
    )

    min_id_cluster = img_kmeans.reduceRegion(
        reducer=ee.Reducer.min(), maxPixels=1e12, bestEffort=False
    )

    # Cria uma lista de IDs de cluster
    lista_id_clusters = ee.List.sequence(
        min_id_cluster.get("cluster"), max_id_cluster.get("cluster")
    )

    # Função para calcular a diferença quadrática para um cluster
    def calcula_diff_quad_cluster(id_cluster):
        # Isola os pixels pertencentes ao cluster atual
        pixels_cluster = roi.mask(img_kmeans.eq(ee.Number(id_cluster).toInt()))

        # Calcula a média dos pixels do cluster
        media_pixels_cluster = pixels_cluster.reduceRegion(
            ee.Reducer.mean(), maxPixels=1e12, bestEffort=False
        )
```

(continues on next page)

(continuação da página anterior)

```
        # Calcula a diferença quadrática
        diff_quad = pixels_cluster.subtract(media_pixels_cluster.toImage()).pow(2)

        # Soma a diferença quadrática sobre todas as bandas
        diff_quad_soma_bandas = diff_quad.reduceRegion(
            ee.Reducer.sum(), maxPixels=1e12, bestEffort=False
        )

        # Retorna a soma das diferenças quadráticas para todas as bandas
        return diff_quad_soma_bandas.values().reduce(ee.Reducer.sum())

    # Mapeia a função sobre a lista de IDs de cluster e reduz os resultados
    soma_wcss = lista_id_clusters.map(calcula_diff_quad_cluster).reduce(
        reducer=ee.Reducer.sum()
    )
    value = soma_wcss.getInfo()
    print(f"\t>>> WCSS: {value:.2f}")
    return value
```

Perfeito, agora vamos iterar sobre uma lista de valores de k, fazer o agrupamento e calcular o WCSS para cada um deles.

[ ]:
```
# TODO: código está bastante lento, mas não consegui fazer funcionar usando map (erro␣
↪429 na API do GEE)

# Gera lista com uma sequ^encia dos números de cluster
numero_clusters = range(2, 15, 1)  # start, stop, step

# Lista para armazenar os resultados
k_values = []  # Lista para armazenar os valores de k
wcss_values = []  # Lista para armazenar os valores WCSS

# Loop para aplicar k-means com diferentes números de clusters
# Obs.: Existem várias informações de tempo aqui, mas isso é só para debug
for k in numero_clusters:
    print(f"Executando k-means com k = {float(k):2.0f}")
    start_time = time.time()  # Captura o tempo de início da iteração

    img = executa_kmeans(k, amostra_treinamento, roi)
    k_values.append(k)
    wcss_values.append(wcss(img))

    # Calcula o tempo e imprime no console
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"\tK-means concluído em {elapsed_time:.2f} s.\n")
```

Plotar o gráfico de MSE para cada valor de k, e identificar o ponto de inflexão, que é o número de clusters ótimo.

[ ]:
```
# Plota o gráfico WCSS
from scipy.optimize import curve_fit
```

(continues on next page)

(continuação da página anterior)

```
import numpy as np


# Definindo a forma da função exponencial decrescente.
def exp_decreasing(x, a, b, c):
    return a * np.exp(-b * x) + c


k_values = np.array(k_values)
wcss_values = np.array(wcss_values)


# Ajustando a função aos dados. popt são os valores otimizados para os par^ametros da␣
↪função (a, b e c)
popt, _ = curve_fit(
    exp_decreasing, k_values, wcss_values, p0=(1, 1e-6, 1)
)  # p0 é o palpite inicial para os valores dos par^ametros
# Usando os par^ametros otimizados para prever y com base em x
y_pred = exp_decreasing(k_values, *popt)
# Imprimindo os par^ametros otimizados
print("Par^ametros otimizados: a =", popt[0], "b =", popt[1], "c =", popt[2])
plt.plot(k_values, y_pred, "r-", label="Ajuste exponencial")

plt.plot(k_values[:], wcss_values, "bx-", label="WCSS")
plt.title("Método do Cotovelo para K-Means Clustering")
plt.xlabel("Número de clusters - k")
plt.ylabel("WCSS")
plt.xlim(xmin=1.8)
plt.grid()
plt.legend()
plt.show()
```

A partir do gráfico acima, podemos analisar visualmente qual o número ideal de clusters, que será o ponto de inflexão da curva.

Alternativa para execução do k-means no GEE

[ ]:
```
imagem_classificada = ee.Algorithms.Image.Segmentation.KMeans(
    image=roi, numClusters=5, uniqueLabels=False
)
```

[ ]:
```
my_map.addLayer(imagem_classificada.randomVisualizer(), {}, "k-means3")
my_map
```

2.3.4 Atividade

[ ]:

[ ]:

2.4 Laboratório 4

Mosaicos e classificação supervisionada de imagens no GEE

Objetivos:

1. Remoção de nuvens
2. Composições e mosaicos de imagens
3. Classificação supervisionada no GEE

[ ]:
```
import ee
import geemap
```

[ ]:
```
# Ref: https://developers.google.com/earth-engine/apidocs/ee-authenticate
# Para inicializar a sessão para execução insira o id do projeto em ee.Initialize().
ee.Authenticate()
ee.Initialize(project='id_projeto')
```

2.4.1 Preparar dados

Filtrar coleção de imagens (nuvens, datas e região)

Dessa vez vamos utilizar uma coleção de imagens ao invés de uma imagem única, isso garantirá maior robustez ao nosso modelo de classificação.

Adicionalmente neste laboratório, diferentemente do anterior, utilizar imagens sem nuvens serão essenciais. Para eliminar as nuvens, captura-se múltiplas imagens da mesma área, remove-se as nuvens e, usando a técnica do mosaico, integram-se todas em uma única imagem. Isso preenche lacunas deixadas pelas nuvens removidas, pois os pixels faltantes em uma imagem são substituídos por pixels válidos de outras. O GEE prioriza o pixel da imagem que está no topo da coleção para compor o mosaico.

Vamos ver como fazer isso.

[ ]:
```
# Essa é uma função padrão do
def mask_s2_clouds(image):
    """Masks clouds in a Sentinel-2 image using the QA band.

    Args:
        image (ee.Image): A Sentinel-2 image.

    Returns:
        ee.Image: A cloud-masked Sentinel-2 image.
    """
    qa = image.select("QA60")
```

(continues on next page)

(continuação da página anterior)

```
    # Bits 10 and 11 are clouds and cirrus, respectively.
    cloud_bit_mask = 1 << 10
    cirrus_bit_mask = 1 << 11

    # Both flags should be set to zero, indicating clear conditions.
    mask = qa.bitwiseAnd(cloud_bit_mask).eq(0).And(qa.bitwiseAnd(cirrus_bit_mask).eq(0))

    return image.updateMask(mask).divide(10000)
```

Para nos ajudar com a análise, vamos filtrar somente as imagens que passam por São Paulo. Para isso definimos um ponto e depois utilizamos o método `filterBounds()` do GEE

[ ]:
```
sao_paulo = ee.Geometry.Point(-46.711, -23.641)
sao_paulo
```

Vamos definir os limites para cobertura de nuvem e filtrar a coleção de imagens. Agora vamos filtrar o período em que queremos trabalhar com as imagens.

Alguns comentários:

- Selecionar um período muito longo pode resultar em uma coleção muito grande e pesada para trabalhar
- Para fins de exemplos vamos cobrir o ano de 2020

A dica aqui é trabalhar com o intervalo de datas e valor de cobertura por nuvens até se obter uma coleção de tamanho suficiente para a composição da imagem de forma apropriada

[ ]:
```
limite_cobertura_nuvem = 5  # máximo de 10% de cobertura de nuvem
data_inicial, data_final = "2018-01-01", "2022-01-30"

dataset = (
    ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED")
    .filterBounds(sao_paulo)
    .filterDate(data_inicial, data_final)
    .filter(ee.Filter.lt("CLOUDY_PIXEL_PERCENTAGE", limite_cobertura_nuvem))
    .map(mask_s2_clouds)
)
dataset
```

Podemos verificar o número de imagens resultantes na coleção com o método `size()`:

[ ]:
```
print(f"total de imagens na coleção: {dataset.size().getInfo()}")
```

Recortar a região de interesse (*Region of Interest - ROI*)

Primeiramente vamos definir um polígino de interesse para nossa análise.

A fins de exemplo, vamos definir programaticamente um polígono retangular que cobre a cidade de São Paulo. Mas tenha em mente que você também pode selecionar no mapa usando a ferramenta de desenho.

[ ]:
```
# roi: região de interesse
roi = ee.Geometry.BBox(
    west=-46.88655, south=-23.6906, east=-46.611642, north=-23.590958
```

(continues on next page)

(continuação da página anterior)

```
)
roi
```

Como no trabalho a ideia é utilizar uma coleção de imagens, o método `clip()` não pode ser aplicado diretamente.

Para efetuar o procedimento uma função personalizada deve ser criada. No caso, abaixo criamos a função `clip_img_collection()`, que recebe uma coleção de imagens e uma geometria (no caso, o nosso polígono de interesse) e retorna a coleção já recortada.

[ ]:
```
def clip_img_collection(dataset, geometria):
    """Recorta todas as imagens de uma coleção de imagens do Earth Engine para
    uma região específica definida por uma geometria.

    Essa função cria internamente uma função de recorte específica para a
    geometria fornecida e aplica essa função a cada imagem na coleção de imagens
    usando o método map().

    Parameters
    ----------
    dataset : ee.ImageCollection
        A coleção de imagens a ser recortada. Cada imagem da coleção será
        recortada para se adequar à geometria fornecida.

    geometria : ee.Geometry
        A geometria que define a região de interesse para onde as imagens serão
        recortadas. A geometria pode ser um ponto, linha, polígono, etc.

    Returns
    -------
    ee.ImageCollection
        Uma nova coleção de imagens com cada imagem recortada para a região
        definida pela geometria fornecida.
    """

    def clip_func(image):
        return image.clip(geometria)

    return dataset.map(clip_func)
```

[ ]:
```
# Faz o clip da região desejada em toda coleção.
clipped_dataset = clip_img_collection(dataset, roi)
clipped_dataset
```

Remoção de valores inválidos

Após a remoção das nuvens, para melhorar ao conjunto de dados, vamos eliminar valores discrepantes.

Sabemos que a reflectância deve ser menor do que 1, então vamos criar uma máscara para eliminar valores maiores do que 1.

As máscaras são geradas a partir do comparador `lt()` (*less than* - menor do que) do GEE.

[ ]:
```
# Função para remoção de valores inválidos
def remove_valores_invalidos(collection, bandas):
    """Aplica uma máscara para remover valores inválidos em uma coleção de
    imagens.

    Os valores considerados inválidos são aqueles maiores que 1 para as bandas
    especificadas.

    Parameters
    ----------
    collection : ee.ImageCollection
        A coleção de imagens a ser processada.
    bandas : list of str
        Lista com os nomes das bandas que serão processadas.

    Returns
    -------
    ee.ImageCollection
        A coleção de imagens com valores inválidos removidos.
    """

    def mask_out_of_range_reflectance(imagem):
        """Mascara valores de reflect^ancia acima de 1"""
        for banda in bandas:
            imagem = imagem.updateMask(imagem.select(banda).lt(1))
        return imagem

    return collection.map(mask_out_of_range_reflectance)
```

[ ]:
```
# Executa a função de remoção dos valores inválidos na coleção.
clipped_dataset = remove_valores_invalidos(clipped_dataset, ["B2", "B3", "B4", "B8"])
clipped_dataset
```

Composição e Mosaico

Para se criar um mosaico a partir de uma coleção de imagens, basta invocar o método `mosaic()` da coleção de interesse.

Já para a criação de uma composição, um método do tipo reducer que faz um cálculo pixel a pixel deve ser chamado a partir do objeto que contém a coleção. Nesse caso, os pixels transparentes são ignorados durante os cálculos.

Os cálculos são feitos de forma separada para cada banda das imagens da coleção, resultando assim em uma imagem com as mesmas bandas da coleção, onde cada pixel de cada banda é preenchido por um valor calculado a partir da banda de mesmo nome na coleção.

[ ]:
```
# Criação de mosaico a partir da coleção.
mosaico = clipped_dataset.mosaic()
# TODO: não vamos usar o mosaico pra nada? Roteiro antigo não ajudou muito.

# Criação de composição a partir da coleção.
composition = clipped_dataset.mean()
```

Vamos conferir o resultado da composição através do método `bandTypes()` que retorna as bandas da imagem.

[ ]:
```
composition.bandTypes()
# TODO: for a strange reason, it is giving me numbers higher than 1.0, needs checking
```

Calcular NDVI

Agora queremos calcular a banda de índice de vegetação (`NDVI`) para cada imagem da coleção. Vamos seguir os mesmos passos do laboratório anterior!

Alguns comentários:

- `B2` é a banda azul
- `B3` é a banda verde
- `B4` é a banda vermelha
- `B8` é a banda infravermelho próximo (NIR)
- `MSK_CLDPRB` é a banda de probabilidade de nuvens
- `SCL` é fruto de um processo de classificação pelo algoritmo de classificação de cena (Scene Classification)

Sempre verifique a documentação oficial: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR_HARMONIZED#bands

[ ]:
```
# calcular a banda NDVI conforme visto no laboratório anterior.

img_bd_nir = composition.select("B8")
img_bd_red = composition.select("B4")

## adicionar a banda NDVI à coleção de imagens
composition = composition.addBands(
    img_bd_nir.subtract(img_bd_red).divide(img_bd_nir.add(img_bd_red)).rename("NDVI"),
    ["NDVI"],
)
composition
```

Seleção de bandas para a classificação

Vamos criar listas com nomes das bandas para utilizarmos ao longo deste laboratório.

[ ]:
```
# Lista com somente as bandas de interesse para a classificação
classification_bands = ["B2", "B3", "B4", "B8", "NDVI"]
```

[ ]:
```
# Descarta as bandas usadas para se eliminar efeitos de nuvens.
composition = composition.select(classification_bands)
composition
```

2.4.2 Classificação supervisionada

Definindo amostras conhecidas

Vamos começar a definir visualmente regiões conhecidas para treinamento do modelo de classificação. Em outras palavras, precisamos gerar amostras sobre a imagem para que o modelo aprenda a classificar os pixels.

Essa é a magia da classificação supervisionada... O ser humano aponta para alguns polígonos e diz para o computador: «olha, isso aqui é água, isso aqui é vegetação, isso aqui é solo exposto, etc». Em seguida, o computador aprende a classificar os pixels da imagem de acordo com o que foi definido pelo ser humano.

Essa é uma etapa tão especial que merece ser apresentada pelo próprio Dr. Qiusheng Wu, criador do `geemap` e Prof. da Universidade do Tennessee, nos EUA. Veja:

[ ]:
```
import geemap

# TODO: the process described in the video is temporally broken
# see: https://github.com/gee-community/geemap/discussions/1816
geemap.show_youtube("https://youtu.be/VWh5PxXPZw0")
```

Vamos definir um dicionário para armazenar os parâmetros de visualização básicos para o mapa.

[ ]:
```
visualization = {
    "min": 0.0,
    "max": 0.3,
    "bands": ["B4", "B3", "B2"],
}
```

Agora de fato começaremos a definir as regiões de treinamento. Devido a um erro no `geemap`, vamos coletar classe a classe e depois juntar tudo em um único `FeatureCollection`.

Você deve seguir os seguintes passos:

1. Execute a célula que começa com `my_map = geemap.Map(`
2. Clique no botão `toolbar` no canto superior direito do mapa
3. Clique no botão `expand tool bar` (sinal de +)
4. Clique no botão `collect training samples` (símbolo de um dedo indicador)
5. Altere o campo `Required` para `classe`
6. Altere o campo `Integer` para um número inteiro de sua escolha. Recomenda-se começar com 0 e ir aumentando de 1 em 1 para cada classe. **É extremamente importante que cada classe receba um número diferente neste campo**, por exemplo, 0=agua, 1=vegetacao, 2=solo exposto, etc.

7. Altere o campo `Optional` para `label`
8. Altere o campo `String` para o nome da classe, por exemplo, `agua`, `vegetacao`, `solo exposto`, etc.
9. Clique no botão `Draw a polygon` (símbolo de um hexágono) no canto esquerdo do mapa
10. Desenhe um polígono sobre a imagem que aparece no mapa.
11. Ao desenhar todos seus polígonos, execute a célula que vem logo em seguida a esta. Não volte a executar a mesma célula novamente, pois isso irá apagar todas as suas amostras de treinamento.

[ ]:
```
# executar esta célula e desenhar polígonos de AGUA
my_map = geemap.Map(lite_mode=False)
my_map.set_center(-46.711, -23.641, 12)
my_map.addLayer(composition, visualization, "mosaico")
my_map
```

[ ]:
```
# executar esta célula somente após desenhar todos os polígonos de AGUA
agua = my_map.user_rois
agua
```

[ ]:
```
# executar esta célula e desenhar polígonos de VEGETACAO RASTEIRA
my_map = geemap.Map(lite_mode=False)
my_map.set_center(-46.711, -23.641, 12)
my_map.addLayer(composition, visualization, "mosaico")
my_map
```

[ ]:
```
# executar esta célula somente após desenhar todos os polígonos de VEGETACAO RASTEIRA
vegeta_baixa = my_map.user_rois
vegeta_baixa
```

[ ]:
```
# executar esta célula e desenhar polígonos de VEGETACAO ALTA
my_map = geemap.Map(lite_mode=False)
my_map.set_center(-46.711, -23.641, 12)
my_map.addLayer(composition, visualization, "mosaico")
my_map
```

[ ]:
```
# executar esta célula somente após desenhar todos os polígonos de VEGETACAO ALTA
vegeta_alta = my_map.user_rois
vegeta_alta
```

[ ]:
```
# executar esta célula e desenhar polígonos de CONSTRUÇÕES
my_map = geemap.Map(lite_mode=False)
my_map.set_center(-46.711, -23.641, 12)
my_map.addLayer(composition, visualization, "mosaico")
my_map
```

[ ]:
```
# executar esta célula somente após desenhar todos os polígonos de CONSTRUÇÕES
construcoes = my_map.user_rois
construcoes
```

[ ]:
```
# executar esta célula e desenhar polígonos de SOLO EXPOSTO
my_map = geemap.Map(lite_mode=False)
```

(continues on next page)

(continuação da página anterior)

```
my_map.set_center(-46.711, -23.641, 12)
my_map.addLayer(composition, visualization, "mosaico")
my_map
```

[ ]:
```
# executar esta célula somente após desenhar todos os polígonos de SOLO EXPOSTO
solo_exposto = my_map.user_rois
solo_exposto
```

Mesclando amostras em uma única feature collection

Os objetos `agua`, `vegeta_baixa`, `vegeta_alta`, `construcoes` e `solo_exposto` são do tipo `ee.FeatureCollection`. Vamos mesclar todos em uma única feature collection para facilitar a manipulação dos dados. Utilizaremos o método `merge()` do GEE.

[ ]:
```
# Criação de um único objeto do tipo ee.FeatureCollection com todos os objetos.
collection = (
    agua.merge(vegeta_baixa).merge(vegeta_alta).merge(construcoes).merge(solo_exposto)
)
collection
```

Sampling

O procedimento anterior apenas delimitou as regiões de onde se deseja coletar amostra, associando-as com suas respectivas classes, e não colheu nenhuma amostra propriamente.

Vamos fazer a coleta dos valores de pixels nas regiões.

Adicionalmente, vamos definir a escala que queremos trabalhar, nesse caso 10 metros pois é o menor valor (mais preciso) disponível para o Sentinel-2.

[ ]:
```
# Ajuste este valor para aumentar ou diminuir a área de amostragem
SCALE = 10  # meters
```

[ ]:
```
# Extrai o valor de todos os pixels nas regiões amostradas.
amostra_treinamento = composition.sampleRegions(
    collection=collection, properties=["classe"], scale=SCALE
)
```

[ ]:
```
amostra_treinamento
```

Treinamento

Para treinar o modelo, basta utilizar um classificador com os parâmetros desejados e utilizar o método `train()`.

Vamos utilizar o classificador Random Forest.

O primeiro parâmetro do `train()` é a amostra a ser utilizada para treinar o modelo, o segundo é a propriedade que contém o número que identifica a classe e o último, a lista com as bandas que serão utilizadas para classificação.

Na instanciação do classificador, `smileRandomForest()`, o parâmetro é a quantidade de árvores de decisão que devem ser criadas. Esse valor pode, e deve, ser ajustado de acordo com os resultados no conjunto de validação.

Outros parâmetros podem ser encontrados na documentação do GEE e a explicação da utilidade dos mesmos em literatura apropriada.

```
[ ]: # Instancia um classificador na memória com os par^ametros dados e treinando no conjunto␣
     ↪de treinamento.
     classificador_treinado = ee.Classifier.smileRandomForest(50).train(
         amostra_treinamento, "classe", classification_bands
     )
     classificador_treinado
```

Um primeiro teste pode ser feito com a própria amostra de treinamento, com a criação da matriz de confusão a partir do método `confusionMatrix()`.

A acurácia provavelmente será altíssima, visto que foi o próprio conjunto utilizado no treinamento que foi avaliado.

Caso o desempenho tenha sido ruim sobre o conjunto de treinamento, é um provável caso de underfitting.

```
[ ]: # // Matriz de confusão do conjunto de treinamento e acurácia.
     matriz_confusao = classificador_treinado.confusionMatrix()
     matriz_confusao
```

```
[ ]: print(
         f"Acurácia no conjunto de treinamento: {matriz_confusao.accuracy().getInfo()*100:.4f}
     ↪%"
     )
```

Classificação

Para classificar a imagem a partir do treinamento, invoca-se o método `classify()` a partir da imagem e com o classificador treinado escolhido como parâmetro.

O resultado é um objeto do tipo imagem com uma única banda em que os pixels armazenam o valor relativo às classes que foram atribuídas no momento do desenho das regiões das amostras em tela.

```
[ ]: # Classifica a imagem com o classificador treinado e com os par^ametros definidos.
     imagem_classificada = composition.classify(classificador_treinado)
     imagem_classificada
```

Visualização

Para visualizar a imagem, primeiro uma paleta de cores deve ser definida.

Essa paleta deve ser feita como uma lista de strings em que cada cor é uma sequência de seis dígitos em hexadecimal.

A ordem em que as cores estão na string representa o número inteiro associado com a classe a partir de 0.

```
[ ]: # Paleta de cores para adicionar a imagem classificada ao mapa.
     paleta_cores_classes = [
         "#1f77b4",  # Água - um azul mais suave e claro.
         "#98df8a",  # Vegetação rasteira - um verde claro para diferenciar da vegetação alta.
         "#2ca02c",  # Vegetação alta - um verde mais vibrante e menos saturado.
         "#7f7f7f",  # Construção - um cinza médio que representa áreas construídas.
         "#ff7f0e",  # Solo exposto - um laranja mais vibrante e atraente.
     ]
```

Finalmente, podemos visualizar a imagem classificada.

[ ]:
```
final_map = geemap.Map(lite_mode=False)
final_map.centerObject(roi, 12)
vis_params = {
    "min": 0,
    "max": 4,
    "bands": ["classification"],
    "palette": paleta_cores_classes,
}
final_map.addLayer(
    imagem_classificada.clip(roi), vis_params, "Imagem Classificada", True
)
final_map
```

2.4.3 Atividade

Este laboratório não requer uma atividade específica. Você pode aproveitar esse tempo para aplicar os conceitos no seu trabalho prático.

[ ]:
```
# Exportar a imagem classificada para um shapefile
geemap.ee_export_vector(
    ee_object=collection,
    filename="collection.shp",
    selectors=None,
    verbose=True,
    keep_zip=True,
    timeout=300,
    proxies=None,
)
```

[ ]:
```
# %pip install pycrs
```

[ ]:
```
geemap.shp_to_ee("collection.shp")
```

2.5 Laboratório 5

Avaliação de classificação supervisionada de imagens no GEE

Objetivos:

1. Divisão de amostras para avaliação de classificação
2. Classificação supervisionada
3. Seleção de modelo

[ ]:
```
import random

import ee
```

(continues on next page)

(continuação da página anterior)

```
import geemap
```

```
[ ]: # Ref: https://developers.google.com/earth-engine/apidocs/ee-authenticate
     # Para inicializar a sessão para execução insira o id do projeto em ee.Initialize().
     ee.Authenticate()
     ee.Initialize(project='id_projeto')
```

2.5.1 Preparação dos dados

Primeiro definimos uma região de interesse, que pode convenientemente ser a mesma área do laboratório anterior, o que facilitará bastante o desenvolvimento deste laboratório.

```
[ ]: # roi: região de interesse
     roi = ee.Geometry.BBox(
         west=-46.88655, south=-23.6906, east=-46.611642, north=-23.590958
     )
     roi
```

Vamos realizar algumas operações idênticas ao laboratório anterior, para carregar a imagem do `Sentinel-2` e preparar os dados para o laboratório.

Para começar, vamos definir algumas variáveis globais que serão utilizadas ao longo do laboratório.

```
[ ]: escala = 10  # resolução espacial, em metros
     limite_nuvens = 10  # limite de cobertura de nuvens, em porcentagem
     dataset = ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED")  # dataset original
     sao_paulo = ee.Geometry.Point([-46.6333, -23.5500])  # um ponto qualquer em São Paulo
     data_inicio = "2019-01-01"
     data_fim = "2023-01-01"
     banda_infra_vermelho = "B8"  # banda do infravermelho próximo (NIR)
     banda_vermelho = "B4"  # banda do vermelho
```

Agora definimos algumas funções relevantes para o pré-processamento dos dados.

```
[ ]: def recorta(img: ee.Image) -> ee.Image:
         """Função para recortar parte da img na região desejada. Assume que já
         existe uma variável roi definida."""
         return img.clip(roi)


     def remove_nuvens(img: ee.Image) -> ee.Image:
         """Função para remover nuvens de uma img. Só funciona para o Sentinel-2."""
         return (
             img.updateMask(img.select("MSK_CLDPRB").lt(limite_nuvens))
             .updateMask(img.select("SCL").neq(3))
             .updateMask(img.select("SCL").neq(7))
             .updateMask(img.select("SCL").neq(8))
             .updateMask(img.select("SCL").neq(9))
             .updateMask(img.select("SCL").neq(10))
         )
```

(continues on next page)

(continuação da página anterior)

```
def remove_valores_invalidos(img: ee.Image) -> ee.Image:
    """Função para remoção de valores inválidos. Funciona somente para o
    Sentinel-2."""
    return (
        img.updateMask(img.select("B2").lt(10000))
        .updateMask(img.select("B3").lt(10000))
        .updateMask(img.select("B4").lt(10000))
        .updateMask(img.select("B8").lt(10000))
    )
```

Agora executamos o pré-processamento dos dados, igual fizemos no laboratório anterior.

```
[ ]: # Filtra o dataset para a região, datas e bandas de interesse.
dataset = (
    dataset.filterBounds(sao_paulo)
    .filterMetadata("CLOUDY_PIXEL_PERCENTAGE", "less_than", limite_nuvens)
    .filterDate(data_inicio, data_fim)
    .select(["B2", "B3", "B4", "B8", "MSK_CLDPRB", "SCL"])
)

# Recorta o dataset para a região de interesse.
dataset_clipped = dataset.map(recorta)

# Remove as nuvens do dataset.
dataset_sem_nuvens = dataset_clipped.map(remove_nuvens)

# Remove valores inválidos na coleção.
dataset_sem_nuvens = dataset_sem_nuvens.map(remove_valores_invalidos)
dataset_sem_nuvens
```

Até aqui apenas editamos o dataset, vamos então criar uma `ee.Image` a partir dele, tal qual fizemos no laboratório anterior.

```
[ ]: # Cria o mosaico a partir da coleção.
imagem = dataset_sem_nuvens.mosaic()

# Cria a composição a partir da coleção.
imagem = dataset_sem_nuvens.mean()

# Ajuste nos valores de reflect^ancia das imagens.
imagem = imagem.multiply(0.0001)

# Extrai bandas do vermelho e do infravermelho próximo
imagem_banda_nir = imagem.select(banda_infra_vermelho)
imagem_banda_vermelho = imagem.select(banda_vermelho)

# Calcula o NDVI
imagem = imagem.addBands(
    imagem_banda_nir.subtract(imagem_banda_vermelho)
    .divide(imagem_banda_nir.add(imagem_banda_vermelho))
```

(continues on next page)

(continuação da página anterior)

```
    .rename("NDVI"),
    ["NDVI"],
)

# Ajuste nos valores de reflect^ancia das imagens.
# imagem = imagem.multiply(0.0001)  # TODO: por que tem que multiplicar 2 vezes?

# Seleciona as bandas de interesse.
imagem = imagem.select(ee.List(["B2", "B3", "B4", "B8", "NDVI"]))
imagem
```

Vamos utilizar os resultados do laboratório anterior para treinar e avaliar um modelo de classificação supervisionada. Para isso, vamos precisar do arquivo shapefile coletado ao final do laboratório anterior. Se você não tiver o arquivo `samples.shp` no seu drive, copie-o do diretório `data` do repositório do curso no GitHub.

O módulo `geopandas` é necessário para ler o arquivo shapefile. Execute a célula abaixo para instalar o módulo, caso necessário.

[ ]:
```
# %pip install geopandas
```

[ ]:
```
# TODO: usar um link pro github, colocar caminho absoluto.
amostras_completas = geemap.shp_to_ee("../data/samples_lab4/samples_lab4.shp")
amostras_completas
```

[ ]:
```
amostras_agua = amostras_completas.filterMetadata("classe", "equals", 0)
amostras_veget_baixa = amostras_completas.filterMetadata("classe", "equals", 1)
amostras_veget_alta = amostras_completas.filterMetadata("classe", "equals", 2)
amostras_construcoes = amostras_completas.filterMetadata("classe", "equals", 3)
amostras_solo = amostras_completas.filterMetadata("classe", "equals", 4)
```

Utilizamos o método `sampleRegions` para extrair amostras de pixels da imagem para cada polígono do shapefile.

Aqui é importante que as `amostras` estejam contidas no mesmo lugar geométrico da `imagem`.

[ ]:
```
# amostras_selecionadas = imagem.sampleRegions(
#     collection=amostras_completas, properties=["classe"], scale=escala
# )
```

[ ]:
```
# amostras_selecionadas
```

2.5.2 Classificação supervisionada

A partir do passo da coleta das amostras vamos prosseguir com algumas opções mais avançadas. Veremos alguns métodos alternativos de seleção de pixels para treinamento e teste.

Amostras com tamanho pré-definido

Começaremos com a segunda opção de amostragem, pegando um número limitado de pixels por classe.

Para prevenir sobrecarga no GEE, vamos limitar o tamanho das amostras por classe usando a função `sample()` em cada `ee.FeatureCollection` do dicionário `amostras_dict`.

A `sample()` é uma função que automatiza a amostragem de pixels numa região, iniciando com uma semente aleatória.

[ ]:
```
def coleta_pixels_amostra(regiao_amostra: ee.Geometry) -> ee.FeatureCollection:
    """Realiza uma amostragem limitada de pixels dentro das regiões de uma
    FeatureCollection."""
    # Gera uma semente aleatória para a amostragem
    semente = ee.Number(random.random()).multiply(10000).toLong()

    # Amostra a imagem dentro da região dada
    sample = imagem.sample(
        region=ee.FeatureCollection(regiao_amostra),
        dropNulls=True,  # Descarta quaisquer amostras com valores nulos
        seed=semente,  # Usa a semente gerada para amostragem aleatória
        scale=escala,  # A escala na qual a amostragem deve ser feita
        numPixels=200,  # O número de pixels para coletar
        geometries=False,  # Falso para economizar tempo de processamento
    )

    # Recupera o número da classe da primeira feição na coleção
    numero_classe = ee.FeatureCollection(regiao_amostra).first().get("classe")

    # Mapeia sobre a amostra, definindo o número da classe para cada feição
    return sample.map(lambda feature: feature.set({"classe": numero_classe}))
```

Em seguida vamos criar uma lista com as feature collections das regiões amostradas para cada classe.

[ ]:
```
lista_amostra_classes = ee.List(
    [
        amostras_agua,
        amostras_veget_baixa,
        amostras_veget_alta,
        amostras_construcoes,
        amostras_solo,
    ]
)
lista_amostra_classes
```

Agora precisamos executar a função para colher amostras nas feature collections de cada classe a partir da lista de amostras. O resultado será uma lista com uma feature collection para cada cada classe com a respectiva amostra.

[ ]:
```
lista_amostra_pixels = lista_amostra_classes.map(coleta_pixels_amostra)
lista_amostra_pixels
```

Finalmente, as Feature Collections são combinadas e achatadas usando `flatten()` para formar uma única Feature Collection com todas as Features.

```
[ ]: amostras_pixels = ee.FeatureCollection(lista_amostra_pixels).flatten()
amostras_pixels
```

2.5.3 Divisão das amostras em subconjuntos

Vamos dividir as amostras em 3 conjuntos: treinamento, validação e testes.

Uma semente aleatória gera números entre 0 e 1 para cada Feature de pixel em `random`. As amostras são divididas usando filtros em `random`, mantendo proporções uniformes. Por exemplo, `random` < 0.7 aloca 70% para treino, e 15% para teste e validação cada.

```
[ ]: semente = ee.Number(random.random()).multiply(10000).toLong()
```

Adiciona uma propriedade a cada pixel da amostra com um número real entre 0 e 1 aleatório.

```
[ ]: amostras_pixels = amostras_pixels.randomColumn("random", semente)
amostras_pixels
```

Vamos separar 70% dos dados para treinamento, 15% para validação e 15% para teste final.

```
[ ]: treinamento = amostras_pixels.filter(ee.Filter.lt("random", 0.7))
validacao = amostras_pixels.filter(ee.Filter.gte("random", 0.7)).filter(
    ee.Filter.lt("random", 0.85)
)
teste = amostras_pixels.filter(ee.Filter.gte("random", 0.85))
```

```
[ ]: treinamento  # para visualizar a amostra de treinamento
```

```
[ ]: validacao  # para visualizar a amostra de validação
```

```
[ ]: teste  # para visualizar a amostra de teste
```

Treinamento e avaliação dos modelos

O passo inicial do processo é treinar os modelos desejados e avaliá-los na amostra de validação, para enfim tomar uma decisão sobre a escolha do modelo, ou reparametrização.

Treinamento

Para treinar o modelo, é necessário criar uma instância do classificador selecionado, configurando-o com os parâmetros apropriados, e depois aplicar o método `train()`.

No nosso caso, usaremos o classificador `smileRandomForest()`, configurado inicialmente para usar 50 árvores de decisão. Este número de árvores é ajustável e deve ser refinado baseado no desempenho observado no conjunto de validação. Informações adicionais sobre outros parâmetros configuráveis podem ser consultadas na documentação do Google Earth Engine (GEE), e suas aplicações práticas estão detalhadas na literatura especializada.

[ ]:
```
# Instancia um classificador na memória com os par^ametros dados e treinando no conjunto␣
↪de treinamento.
classificador_treinado = ee.Classifier.smileRandomForest(50).train(
    features=treinamento,  # amostra a ser usada para treinamento
    classProperty="classe",  # propriedade que contém o número que identifica a classe
    inputProperties=ee.List(["B2", "B3", "B4", "B8", "NDVI"]),  # lista de bandas
)
classificador_treinado
```

Validação

O processo de validação tenta compensar os efeitos do conjunto de treinamento sobre a matriz de confusão. Aqui, o classificador treinado é usado para classificar a amostra de validação e uma matriz de erros é gerada.

A tendência é que a acurácia seja menor, caso a acurácia reduza excessivamente, pode ser efeito de um *overfitting*, onde o modelo tem um desempenho excelente no conjunto de treinamento, mas no de validação tem um desempenho ruim. Nesse caso, ajustes devem ser feitos nos parâmetros do modelo e conjunto de treinamento.

/vamos iniciar aplicando o classificador que foi treinado no conjunto de treinamento sobre o conjunto de validação.

[ ]:
```
validacao_classificada = validacao.classify(classificador_treinado)
validacao_classificada
```

Calcula a matriz de erros do classificador aplicado ao conjunto de validação

[ ]:
```
matriz_validacao = validacao_classificada.errorMatrix("classe", "classification")
matriz_validacao
```

Calcula a acurácia do classificador aplicado ao conjunto de validação

[ ]:
```
matriz_validacao.accuracy()
```

Teste final e classificação da imagem

Vamos classificar o conjunto de testes usando o modelo validado, aplicando o método `classify()` do GEE ao objeto da amostra de testes com o classificador treinado.

[ ]:
```
validacao_classificada = validacao.classify(classificador_treinado)
validacao_classificada
```

A matriz de erro é então calculada e podemos visualizar a acurácia dos testes.

[ ]:
```
matriz_validacao = validacao_classificada.errorMatrix("classe", "classification")
matriz_validacao
```

Aqui é feita a avaliação do modelo escolhido, estimando seu erro de predição em novos dados através da acurácia.

[ ]:
```
matriz_validacao.accuracy()
```

O resultado do método `classify()`, diferentemente dos conjuntos de validação e teste, é um objeto do tipo `ee.Image` com uma única banda em que os pixels armazenam o valor relativo às classes que foram atribuídas no momento do desenho das regiões das amostras em tela.

Abaixo vamos aplicar o classificador que foi treinado no conjunto de treinamento sobre o conjunto de teste.

[ ]:
```
teste_classificada = teste.classify(classificador_treinado)
teste_classificada
```

Calcula a matriz de erros da amostra de testes.

[ ]:
```
matriz_teste = teste_classificada.errorMatrix("classe", "classification")
matriz_teste
```

Calcula a acurácia da matriz de erros da amostra de testes.

[ ]:
```
matriz_teste.accuracy()
```

Classifica a imagem com o classificador treinado e com os parâmetros definidos.

[ ]:
```
imagem_classificada = imagem.classify(classificador_treinado)
imagem_classificada
```

Visualiza o mapa final com as imagens geradas

[ ]:
```
paleta_cores = [
    "#1f77b4",  # Água - um azul mais suave e claro.
    "#98df8a",  # Vegetação rasteira - um verde claro para diferenciar da vegetação alta.
    "#2ca02c",  # Vegetação alta - um verde mais vibrante e menos saturado.
    "#7f7f7f",  # Construção - um cinza médio que representa áreas construídas.
    "#ff7f0e",  # Solo exposto - um laranja mais vibrante e atraente.
]
```

[ ]:
```
my_map = geemap.Map()
my_map.centerObject(roi, 12)
my_map.addLayer(
    imagem_classificada, {"min": 0, "max": 4, "palette": paleta_cores}, "Classificação"
)
my_map
```

CAPÍTULO 3

Referências Bibliográficas

Bibliografia

[ARTHUR2006] ARTHUR, D.; VASSILVITSKII, S. *k-means++: The advantages of careful seeding*. Stanford, 2006.

[BOEHMKE2019] BOEHMKE, B.; GREENWELL, B. M. *Hands-on machine learning with R*. CRC Press, 2019. Disponível em: https://bradleyboehmke.github.io/HOML/. Acesso em: 8 jun. 2021.

[CROSTA1992] CRÓSTA, A. P. *Processamento Digital de Imagens de Sensoriamento Remoto*. Campinas: Instituto de Geociências, UNICAMP, 1992.

[FRIEDMAN2001] FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. New York: Springer series in statistics, 2001.

[GARETH2013] GARETH, J. et al. *An introduction to statistical learning: with applications in R*. Spinger, 2013. Disponível em: https://www.statlearning.com/. Acesso em: 8 jun. 2021.

[GINCIENE2011] GINCIENE, B. R.; BITENCOURT, M. D. *Utilização do EVI (Enhanced Vegetation Index) para maior sensibilidade na detecção de mudanças temporais em fragmentos de floresta estacional semidecidual*. Simpósio Brasileiro de Sensoriamento Remoto, Curitiba, PR, 2011.

[GORELICK2017] GORELICK, N.; HANCHER, M.; DIXON, M.; ILYUSHCHENKO, S.; THAU, D.; MOORE, R. *Google Earth Engine: Planetary-scale geospatial analysis for everyone*. Remote Sensing of Environment, v. 202, 2017, p. 18-27. ISSN 0034-4257. Disponível em: https://www.sciencedirect.com/science/article/pii/S0034425717302900. Acesso em: [data de acesso].

[GRIEBLER2011] GRIEBLER, D. *Padrões e Frameworks de Programação Paralela em Arquiteturas Multi-Core*. Porto Alegre: PUCRS, 2011. Disponível em: http://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr064.pdf. Acesso em: 17 ago. 2020.

[LILLESAND2015] LILLESAND, T.; KIEFER, R. W.; CHIPMAN, J. *Remote sensing and image interpretation*. John Wiley & Sons, 2015.

[MILOSLAVSKAYA2016] MILOSLAVSKAYA, Natalia; TOLSTOY, Alexander. *Big data, fast data and data lake concepts*. Procedia Computer Science, v. 88, p. 300-305, 2016.

[NOVO2008] NOVO, E. M. L. M. Sensoriamento Remoto: Princípios e Aplicações. 3rd ed. Edgard Blücher, 2008. Disponível em: http://www.dpi.inpe.br/labisa/livro/res/conteudo.pdf

[OLIVEIRA2019] OLIVEIRA, C. B. de S.; CANDEIAS, A. L. B.; TAVARES, J. R. *Utilização de índices físicos a partir de imagens OLI–TIRS para o mapeamento de uso e cobertura da terra no entorno do aeroporto internacional do Recife/Guararapes–Gilberto Freire*. Revista Brasileira de Geografia Física, v. 12, n. 03, p. 1039-1053, 2019.

[TUCKER1979] TUCKER, C. J. *Red and photographic infrared linear combinations for monitoring vegetation*. Remote Sensing of Environment, v. 8, n. 2, p. 127-150, 1979.