# LABORATÓRIOS DIDÁTICOS DE GEOPROCESSAMENTO

## Sensoriamento Remoto

Avaliação de classificação supervisionada de imagens no GEE

**Docente:** Prof.ª Dr.ª Mariana Abrantes Giannotti

**Elaboração do roteiro:** Leonardo Alves Godoy

# 1 Introdução

Neste laboratório são apresentados uma forma para se efetuar a divisão de amostras para avaliação de classificação, os conceitos dessa avaliação de classificação supervisionada, a seleção de modelo e sua implementação no GEE.

# 2 Referências teóricas

## 2.1 Avaliação e seleção de modelo

Para poder se 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.

A seleção de modelo tem por objetivo checar a performance de diferentes modelos para se escolher o melhor. Já a avaliação de um modelo é uma forma de, dado um modelo escolhido, estimar seu erro de predição sobre novos dados (FRIEDMAN; HASTIE; TIBSHIRANI, 2019).

Para se executar as duas tarefas citadas no parágrafo anterior, uma forma adequada é se definir a amostra apresentada e dividi-la em três partes: uma parte para treinamento do modelo (*Training Dataset*), outra chamada de conjunto de validação (*Validation Dataset*) e finalmente uma que é conhecida por conjunto de teste (*Test Dataset*). Normalmente, esses subconjuntos da amostra são separados na proporção de 70% para o de treinamento, 15% para o de validação e 15% para o de teste. Não há uma regra fixa para essas proporções, mas é uma prática comum no meio da ciência de dados.

### 2.1.1 Descrição dos conjuntos utilizados no processo de aprendizado supervisionado

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.

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. A avaliação pode se tornar mais enviesada quando mais características derivadas do conjunto de validação são incorporadas na parametrização do modelo. 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. 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.

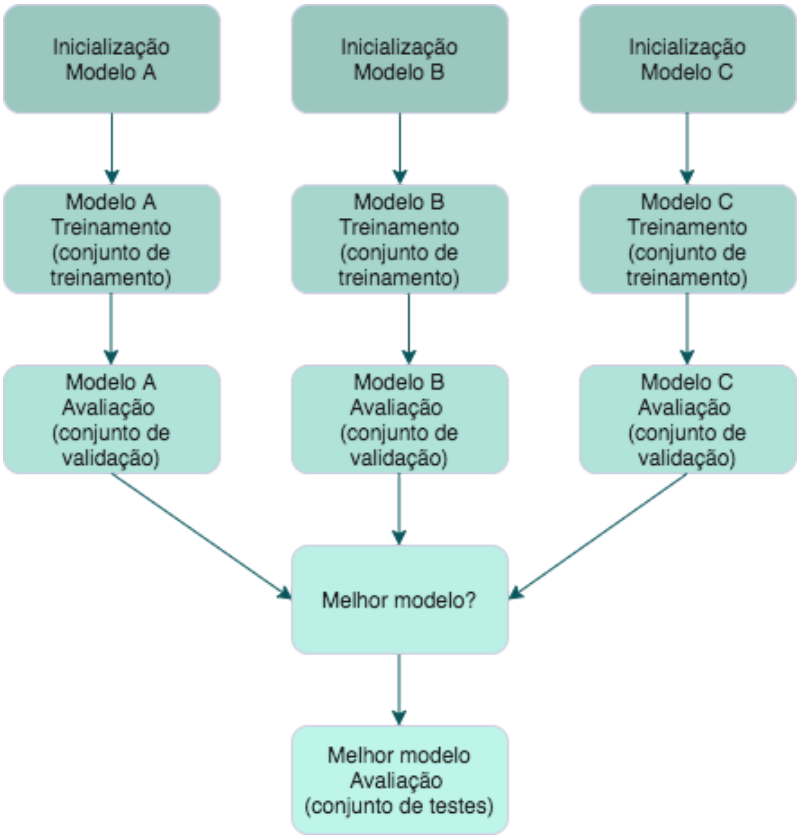
O conjunto de teste é usado para fornecer uma avaliação não enviesada do modelo final e seus parâmetros escolhidos a partir dos procedimentos feitos com o conjunto de validação. Este conjunto é utilizado apenas uma vez no fim do processo, quando o modelo foi treinado com o conjunto de treinamento e validado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

### 2.1.2 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.

Figura 1 - Visualização do procedimento para escolha de modelo



## 3. Roteiro prático

### 3.1 Preparação dos dados

A preparação dos dados para este laboratório deve seguir os mesmos passos do laboratório anterior, de classificação supervisionada. Para agilizar o processo, utilize como base o *script* do laboratório anterior para carregar os dados da coleção ***"Sentinel-2 MSI: MultiSpectral Instrument, Level-2A"*** (ou outra, caso deseje explorar), os parâmetros iniciais - como escala e lista com nome de bandas, fazer a seleção da parte de interesse da coleção, o recorte da área de interesse, tratar as áreas cobertas por nuvens, remover os valores inválidos, fazer o mosaico a partir dos resultados, calcular o *NDVI*, selecionar as bandas utilizadas para a classificação. As amostras utilizadas podem ser copiadas do lab anterior ou refeitas caso desejem treinar para o trabalho prático. Depois de obter as amostras, agregue-as em uma única *Feature Collection* e extraia os valores de todos os pixels das regiões amostradas.

Do Laboratório anterior, temos que todo o início do código descrito acima é apresentado a seguir e serve de gabarito para a sequência deste laboratório:

```
// Centraliza o mapa em São Paulo.
Map.centerObject(retanguloSelecao, 12);

// Ajusta a escala (resolução) que será utilizada para o processamento.
var escala = 10;

//Seleção das bandas a serem trabalhadas
var bandas =ee.List(['B2','B3','B4','B8','MSK_CLDPRB','SCL']);

//Seleção para classificação
var bandasClassificacao =ee.List(['B2','B3','B4','B8','NDVI']);

//Seleção da imagem a ser trabalhada
var colecaoSentinel2Selecao = colecaoSentinel2.filterBounds(saoPaulo)
                                    .filterMetadata('CLOUDY_PIXEL_PERCENTAGE',
'less_than', 90)
                                    .filterDate('2019-01-01','2020-01-01')
                                    .select(bandas);


// Função para recortar parte da imagem na região desejada.
```

```
var clipColecao = function (imagem){
  return(imagem.clip(retanguloSelecao));
};

// Faz o clip da região desejada em toda coleção.
colecaoSentinel2Selecao = colecaoSentinel2Selecao.map(clipColecao);


// Faz a remoção de regiões que possivelmente possuem nuvens.
var removeNuvensColecao = function (imagem){
  var imagemSemNuvens = imagem.updateMask(imagem.select('MSK_CLDPRB').lt(80))
                              .updateMask(imagem.select('SCL').neq(3))
                              .updateMask(imagem.select('SCL').neq(7))
                              .updateMask(imagem.select('SCL').neq(8))
                              .updateMask(imagem.select('SCL').neq(9))
                              .updateMask(imagem.select('SCL').neq(10));
  return imagemSemNuvens;
};

// Executa a função de remoção das nuvens na coleção.
var colecaoSentinel2SelecaoSemNuvens =
colecaoSentinel2Selecao.map(removeNuvensColecao);

// Função para remoção de valores inválidos.
var removeValoresInvalidos = function (imagem){
  var imagemSemValoresInvalidos =
imagem.updateMask(imagem.select('B2').lt(10000))

.updateMask(imagem.select('B3').lt(10000))

.updateMask(imagem.select('B4').lt(10000))

.updateMask(imagem.select('B8').lt(10000));
  return imagemSemValoresInvalidos;
};

// Executa a função de remoção dos valores inválidos na coleção.
var colecaoSentinel2SelecaoSemNuvens =
colecaoSentinel2SelecaoSemNuvens.map(removeValoresInvalidos);

// Criação de mosaico a partir da coleção.
var imagem = colecaoSentinel2SelecaoSemNuvens.mosaic();
// Criação de composição a partir da coleção.
var imagem = colecaoSentinel2SelecaoSemNuvens.mean();

// Ajuste nos valores de reflectância das imagens.
imagem = imagem.multiply(0.0001);

// Extraindo as bandas do vermelho e do infravermelho próximo
var imagemBandaNir = imagem.select('B8');
```

```
var imagemBandaVermelho = imagem.select('B4');

// Código para cálculo do NDVI no GEE
imagem = imagem.addBands(
imagemBandaNir.subtract(imagemBandaVermelho)
.divide(imagemBandaNir.add(imagemBandaVermelho))
.rename('NDVI'),['NDVI']);

//Adicionando imagem NDVI
Map.addLayer(imagem, {bands: ['NDVI'], min: -1, max: 1}, 'NDVI');


// Ajuste nos valores de reflectância das imagens.
imagem = imagem.multiply(0.0001);

// Descarta as bandas usadas para se eliminar efeitos de nuvens.
imagem = imagem.select(bandasClassificacao);

// Agrega todas as amostras coletadas pelo usuário em uma FeatureCollection.
var amostraCompletaGeometrias = amostraAgua.merge(amostraVegetacaoAlta)
                                           .merge(amostraSoloExposto)
                                           .merge(amostraVegetacaoRasteira)
                                           .merge(amostraConstrucoes);

// Extrai o valor de todos os pixels nas regiões amostradas.
var amostraTreinamento = imagem.sampleRegions({
  collection: amostraCompletaGeometrias,
  properties: ['classe'],
  scale: escala
});
```

### 3.2 Classificação supervisionada no GEE

A partir do passo da coleta das amostras apresentado no laboratório anterior, iremos prosseguir com algumas opções mais avançadas.

#### 3.2.1 Amostras com tamanho pré-definido

Muitas vezes, o número de *pixels* coletados de todas as regiões pode ser excessivo, causando lentidão e até mesmo falha no processamento dos algoritmos do GEE. Para lidar com isso, pode ser interessante limitar o tamanho das subamostras relativas a cada classe. Nesse caso, cada *Feature Collection* com as geometrias desenhadas para uma classe é usada de forma separada para se obter uma amostra de *pixels* com um tamanho determinado. Isso é feito com a função ***"sample()"*** chamada a partir da imagem instanciada. Na Figura 2, uma variável com o tamanho fixo das amostras é instanciada, para melhor compreensão do código apresentado na Figura 3.

Figura 2 - Tamanho das amostras

```
96  var tamanhoAmostraClasse = 200;
```

```
// Tamanho da amostra a ser coletada para cada classe.
var tamanhoAmostraClasse = 200;
```

O código da Figura 3 apresenta uma função que faz automaticamente o processo de amostragem dos *pixels* a partir de uma dada região. Essa forma é sugerida, para evitar redundância do código para cada coleção de regiões. A primeira instrução da função gera uma semente aleatória para alimentar o sorteio dos *pixels* nas regiões. A função ***"Math.random()"*** cria um número pseudo-aleatório no intervalo [0, 1[, que a cada chamada é diferente. Esse valor é então encapsulado num objeto do tipo ***"ee.Number"*** para poder ser usado pelo lado do servidor do GEE. O resultado é então multiplicado por 10000 e convertido para um tipo inteiro longo (.toLong()), que é o tipo de uma semente esperada pelo método ***"sample()"*** que será utilizado para coletar as amostras das áreas.

No método ***"sample()"***, as variáveis de entrada devem ser passadas como um dicionário de dados. O dicionário de dados passado, então, como parâmetro, possui:

- a região onde deve ser feita a amostragem em ***"region"***, que no caso é o parâmetro da função ***"coletaPixelsAmostra"***. Um transformação do tipo de dado é feita em ***"regiaoAmostra"*** para garantir que o objeto seja tratado como uma *Feature Collection*, utilizando a chamada ***"ee.FeatureCollection"*** com o objeto ***"regiaoAmostra"*** como parâmetro. Essa transformação de tipo de dado também pode ser denominado *cast*.
- O parâmetro ***"dropNulls"*** elimina as amostras que tiverem valores nulos em suas propriedades.
- Em ***"seed"*** é inserida a semente criada anteriormente e, como ela é sempre instanciada com um número distinto, amostras diferentes serão criadas a cada chamada para uma mesma região (garantindo que haverá aleatoriedade das amostragens).
- Em ***"numPixels"*** é determinado o tamanho da amostra
- O parâmetro ***"geometries"*** pode ser marcado como *true* caso se deseje que a posição do *pixel* seja copiada juntamente com os valores das bandas. Esse parâmetro é útil em caso de visualizações, mas aqui é colocado como ***"false"***, pois somente aumentaria o tempo de processamento sem ter função prática para sequência do exercício.

Diferentemente do uso do ***"sampleRegions()"***, em que a propriedade ***"classe"*** pode ser copiada automaticamente, é necessário inserir a propriedade com o número da classe através de código. Para isso, primeiro se obtém o número da classe usando o método ***"get()"*** - com o nome da propriedade ***"classe"*** como parâmetro - na primeira imagem da *Feature Collection* (usando para isso o método ***"first()"***). O *cast* novamente é feito para garantir a correta interpretação do GEE sobre o objeto ***"regiaoAmostra"***. Para finalizar a função ***"coletaPixelsAmostra"*** é retornada uma *Feature Collection* com os elementos sendo as *Features* com os

valores numéricos de cada *pixel* em propriedades nomeadas como as bandas, além disso, todas as amostras recebem a propriedade **"classe"** com o valor coletado anteriormente em ***"numeroClasse"***. Como cada *Feature* deve ter essa propriedade individualmente, um *map* é feito diretamente em uma função. Essa é uma maneira avançada de se codificar no GEE, mas que torna muito flexível a construção de códigos mais avançados.

Como referência para a coleta de amostras use como base as amostras coletadas por Marcos Rosa, disponível no seguinte código que foi retrabalhado por alunos em semestres anteriores:

https://code.earthengine.google.com/002562eb88be5303c1ad5ed8494c9c9e.

Com a função ***"coletaPixelsAmostra"*** construída, basta criar uma lista com todas as *Feature Collections* desenhadas no mapa, como na Figura 4 e utilizar a mesma para se fazer um *map* em cada uma das *Feature Collections* dentro da lista - Figura 5. O último passo é converter a lista resultante, contendo as *Feature Collections* com os valores dos *pixels*, em uma *Feature Collection* usando uma nova conversão ou *cast* (Figura 6). O resultado vai ser uma *Feature Collection* composta de *Feature Collections*. A seguir, basta usar o método ***"flatten()"*** para extrair as *Features* de dentro das *Features Collection* das amostras dos *pixels* e criar uma única *Feature Collection* com todas as *Features* - Figura 6.

Figura 3 - Função para automaticamente colher as amostras das regiões de uma Feature Collection

```
// Função para fazer uma amostra dentro das regiões de uma feature collection.
var coletaPixelsAmostra = function (regiaoAmostra) {
  var semente = ee.Number(Math.random()).multiply(10000).toLong();
  var amostra = imagem.sample({
                                region:ee.FeatureCollection(regiaoAmostra),
                                dropNulls: true,
                                seed: semente,
                                scale:escala,
                                numPixels: tamanhoAmostraClasse,
                                geometries: false
                              });
  var numeroClasse = ee.FeatureCollection(regiaoAmostra).first().get('classe');
  return amostra.map(function (feature) {
                                        return feature.set(
                                                          {'classe': numeroClasse}
                                                         );
                                      }
                     );
};
```

```
// Segunda opção de amostragem, pegando um número limitado de pixels por classe.
// Tamanho da amostra a ser coletada para cada classe.
var tamanhoAmostraClasse = 200;
// Função para fazer uma amostra dentro das regiões de uma feature collection.
var coletaPixelsAmostra = function (regiaoAmostra) {
  var semente = ee.Number(Math.random()).multiply(10000).toLong();
  var amostra = imagem.sample({
                                region:ee.FeatureCollection(regiaoAmostra),
                                dropNulls: true,
                                seed: semente,
                                scale:escala,
                                numPixels: tamanhoAmostraClasse,
                                geometries: false
                              });
  var numeroClasse = ee.FeatureCollection(regiaoAmostra).first().get('classe');
  return amostra.map(function (feature) {
                                        return feature.set(
                                                          {'classe':
numeroClasse}
                                                         );
                                      }
                     );
};
```

Figura 4 - Lista com as regiões coletadas

```
var listaAmostraClasses = ee.List(
                                  [
                                    amostraAgua,
                                    amostraVegetacaoAlta,
                                    amostraSoloExposto,
                                    amostraVegetacaoRasteira,
                                    amostraConstrucoes
                                  ]
                                );
```

```
// Cria uma lista com as feature collections das regiões amostradas para cada
classe.
var listaAmostraClasses = ee.List(
                          [
                            amostraAgua,
                            amostraVegetacaoAlta,
                            amostraSoloExposto,
                            amostraVegetacaoRasteira,
                            amostraConstrucoes
                          ]
                        );
```

Figura 5 - Executa a função para todas as regiões na lista

```
131  var listaAmostraPixels = listaAmostraClasses.map(coletaPixelsAmostra);
```

```
// Executa a função para colher amostras nas feature collections de cada classe
a partir
// da lista com as amostras. O resultado é uma lista com uma feature collection
para
// cada cada classe com a respectiva amostra.
var listaAmostraPixels = listaAmostraClasses.map(coletaPixelsAmostra);
```

Figura 6 - Criação da Feature Collection com os valores dos pixels

```
134  var amostraPixels = ee.FeatureCollection(listaAmostraPixels).flatten();
```

```
// Agrega todas as amostras coletadas pelo usuário em uma FeatureCollection.
var amostraCompletaPixels= ee.FeatureCollection(listaAmostraPixels).flatten();
```

### 3.2.2 Divisão das amostras em subconjuntos

Como já mencionado, é desejável dividir as amostras em conjuntos para treinamento, validação do modelo e testes. A partir da escolha de uma das opções das amostras anteriores, deve ser feito algo similar ao da Figura 31. Primeiro uma semente é criada e em seguida uma propriedade com números aleatórios no intervalo [0, 1[, nomeada ***"random"***, é criada em cada uma das *Features* representando os *pixels*. Em seguida, conforme a Figura 32, são separadas as amostras com filtros aplicados na propriedade ***"random"***. Como os números em ***"random"*** são uniformemente distribuídos, cada parte filtrada deve conter uma

proporção aproximada de acordo com o filtro. Por exemplo, na primeira linha, todas as *Features* com ***"random"*** menor do que 0.7 são selecionadas e atribuídas à ***"amostraTreinamento"***, o que separa em torno de 70% das amostras (Filter.lt ~Lower Than, Filter.gte ~ Greater Than). De forma similar são separados os conjuntos de teste e validação na proporção de 15% da amostra total cada.

Figura 7 - Inserção de propriedade com número aleatório nas *Features*

```
var semente = ee.Number(Math.random()).multiply(10000).toLong();
var amostraCompletaPixels = amostraCompletaPixels.randomColumn('random', semente);
```

```
// Adiciona uma propriedade a cada pixel da amostra com um número real entre 0 e
1 aleatório.
var semente = ee.Number(Math.random()).multiply(10000).toLong();
var amostraCompletaPixels = amostraCompletaPixels.randomColumn('random',
semente);
```

Figura 32 - Separação das subamostras

```
var amostraTreinamento = amostraCompletaPixels.filter(ee.Filter.lt('random', 0.7));
var amostraValidacao = amostraCompletaPixels.filter(ee.Filter.gte('random', 0.7))
                                            .filter(ee.Filter.lt('random', 0.85));
var amostraTeste = amostraCompletaPixels.filter(ee.Filter.gte('random', 0.85));
```

```
// Separa cerca de 70% dos dados para treinamento, utilizando a propriedade com
número aleatório gerada.
// De forma similar, seleciona 15% para validação e 15% para teste final.
var amostraTreinamento = amostraCompletaPixels.filter(ee.Filter.lt('random',
0.7));
var amostraValidacao = amostraCompletaPixels.filter(ee.Filter.gte('random',
0.7))
                                            .filter(ee.Filter.lt('random',
0.85));
var amostraTeste = amostraCompletaPixels.filter(ee.Filter.gte('random', 0.85));
```

### 3.3 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.

### 3.3.1 Treinamento

Para se treinar o modelo, basta instanciar um classificador escolhido com os parâmetros desejados e utilizar o método ***"train()"***. Na Figura 8 é apresentada essa execução com o algoritmo *Random Forest*. O primeiro parâmetro de ***"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. Já 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 na bibliografia.

Figura 8 - Execução do *Random Forest* no GEE

```
161  var classificadorTreinado = ee.Classifier.smileRandomForest(50)
162                                          .train(amostraTreinamento, 'classe', bandasClassificacao);
```

```
// Instancia um classificador na memória com os parâmetros dados e treinando no
conjunto de treinamento.
var classificadorTreinado = ee.Classifier.smileRandomForest(50)
                                          .train(amostraTreinamento, 'classe',
bandasClassificacao);
```

Um primeiro teste pode ser feito com a própria amostra de treinamento, com a criação da matriz de confusão a partir do método *confusionMatrix()* - Figura 9. 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*. O que significa que o treinamento não foi suficiente para se identificar adequadamente as classes. Isso pode ocorrer devido aos parâmetros do modelo, insuficiência ou má distribuição das amostras. Deve ser ajustado antes de se partir para uma validação.

Figura 9 - Matriz de confusão

```
165  var matrizConfusaoAmostraTreinamento = classificadorTreinado.confusionMatrix();
166  print('Matriz de confusão: ', matrizConfusaoAmostraTreinamento);
167  print('Acurácia no conjunto de treinamento: ', matrizConfusaoAmostraTreinamento.accuracy());
```

```
// Matriz de confusão do conjunto de treinamento e acurácia.
var matrizConfusaoAmostraTreinamento = classificadorTreinado.confusionMatrix();
print('Matriz de confusão: ', matrizConfusaoAmostraTreinamento);
print('Acurácia no conjunto de treinamento: ',
matrizConfusaoAmostraTreinamento.accuracy());
```

### 3.3.2 Validação

O processo de validação, tenta compensar os efeitos do conjunto de treinamento na matriz de confusão. Aqui, o classificador treinado é usado para classificar a amostra de validação (Figura 10) e uma matriz de erros é gerada conforme o código apresentado na Figura 11. Nesse caso, a matriz é criada com o método ***"errorMatrix()"*** e usa-se como parâmetro a propriedade ***"classe"*** da amostra e a ***"classification***", resultante da classificação feita. A tendência é que a acurácia seja menor, caso a acurácia reduza muito, pode ser efeito de um *overfitting*, onde o modelo tem uma performance 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. É aqui, também, que quando se está analisando mais de um classificador é feita a tomada de decisão sobre qual será aplicado no conjunto inteiro (imagem). A comparação deve ser feita nas acurácias geradas para cada modelo e, em seguida, selecionado o de melhor performance.

Figura 10 - Classificação da amostra de validação

```
170 var amostraValidacaoClassificada = amostraValidacao.classify(classificadorTreinado);
```

```
// Aplica o classificador que foi treinado no conjunto de treinamento sobre o
conjunto de validação.
var amostraValidacaoClassificada =
amostraValidacao.classify(classificadorTreinado);
```

Figura 11 - Matriz de erro e acurácia da amostra de validação

```
173 var matrizErroAmostraValidacao = amostraValidacaoClassificada.errorMatrix('classe', 'classification');
174 print('Matriz de erros no conjunto de validação: ', matrizErroAmostraValidacao);
175 print('Acurácia no conjunto de validação: ', matrizErroAmostraValidacao.accuracy());
```

```
// Matriz de erros do classificador aplicado ao conjunto de validação e acurácia
da classificação no mesmo.
var matrizErroAmostraValidacao =
```

```
amostraValidacaoClassificada.errorMatrix('classe', 'classification');
print('Matriz de erros no conjunto de validação: ', matrizErroAmostraValidacao);
print('Acurácia no conjunto de validação: ',
matrizErroAmostraValidacao.accuracy());
```

### 3.3.3 Teste final e classificação da imagem

O conjunto de testes deve ter sua classificação feita apenas no modelo que foi escolhido na validação. No GEE, esse processo é feito de forma similar ao do conjunto de validação (Figura 10). Invoca-se o método ***"classify()"*** a partir do objeto contendo a amostra de testes e com o classificador treinado escolhido como parâmetro. A matriz de erro é então calculada de forma similar, como na Figura 12. Para visualizar a acurácia e a matriz de erros do conjunto de testes, basta adaptar o código da Figura 11 para, no caso exemplificado, ***"matrizErroAmostraTeste"***. Aqui é feita a avaliação do modelo escolhido, estimando seu erro de predição em novos dados através da acurácia.

Figura 12 - Matriz de erro do conjunto de testes

```
// Aplica o classificador treinado no conjunto de treinamento no conjunto de validação.
var amostraValidacaoClassificada = amostraValidacao.classify(classificadorTreinado);

// Matriz de erros do classificador aplicado ao conjunto de validação e acurácia da classificação no mesmo.
var matrizErroAmostraValidacao = amostraValidacaoClassificada.errorMatrix('classe', 'classification');
print('Matriz de erros no conjunto de validação: ', matrizErroAmostraValidacao);
print('Acurácia no conjunto de validação: ', matrizErroAmostraValidacao.accuracy());
```

```
// Aplica o classificador que foi treinado no conjunto de treinamento sobre o conjunto de validação.
var amostraValidacaoClassificada = amostraValidacao.classify(classificadorTreinado);

// Matriz de erros do classificador aplicado ao conjunto de validação e acurácia da classificação no
mesmo.
var matrizErroAmostraValidacao = amostraValidacaoClassificada.errorMatrix('classe',
'classification');
print('Matriz de erros no conjunto de validação: ', matrizErroAmostraValidacao);
print('Acurácia no conjunto de validação: ', matrizErroAmostraValidacao.accuracy());
```

A classificação da imagem é feita de forma simples com o método ***"classify()"*** invocado a partir da imagem - Figura 13. O resultado, diferentemente dos conjuntos de validação e teste, 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. As bandas são comparadas pelo nome com o classificador criado.

Figura 13 - Classificação da imagem

```
181  var matrizErroAmostraTeste = amostraTesteClassificada.errorMatrix('classe', 'classification');
// Aplica o classificador treinado no conjunto de treinamento no conjunto de teste.
var amostraTesteClassificada = amostraTeste.classify(classificadorTreinado);

// Calcula a matriz de erros da amostra de testes.
var matrizErroAmostraTeste = amostraTesteClassificada.errorMatrix('classe', 'classification');
print('Matriz de erro no conjunto de teste: ', matrizErroAmostraTeste);
print('Acurácia no conjunto de teste: ', matrizErroAmostraTeste.accuracy());

// Classifica a imagem com o classificador treinado e com os parâmetros definidos.
var imagemClassificada = imagem.classify(classificadorTreinado);
```


```
// Aplica o classificador que foi treinado no conjunto de treinamento sobre o conjunto de teste.
var amostraTesteClassificada = amostraTeste.classify(classificadorTreinado);

// Calcula a matriz de erros da amostra de testes.
var matrizErroAmostraTeste = amostraTesteClassificada.errorMatrix('classe', 'classification');
print('Matriz de erro no conjunto de teste: ', matrizErroAmostraTeste);
print('Acurácia no conjunto de teste: ', matrizErroAmostraTeste.accuracy());

// Classifica a imagem com o classificador treinado e com os parâmetros definidos.
var imagemClassificada = imagem.classify(classificadorTreinado);
```


### 3.4 Visualização da classificação

Para visualizar a imagem, utilize códigos similares aos apresentados ao final do roteiro anterior.

## 4. Referências bibliográficas


BOEHMKE, B.; GREENWELL, B. M. Hands-on machine learning with R. CRC Press, 2019. Disponível em: https://bradleyboehmke.github.io/HOML/. Acesso em: 24 set. 2020.

CRÓSTA, A. P. Processamento Digital de Imagens de Sensoriamento Remoto. Campinas: Ed. Rev. Campinas, UNICAMP. 1992.

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

GORELICK, N. et al. Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, [s. l.], v. 202, p. 18-27. 2017.

GARETH, J. et al. An introduction to statistical learning: with applications in R. Spinger, 2013.

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.

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

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.