De Analista para Analista: Usando o Sits na Prática

Um guia acessível para séries temporais com R

Equipe Mata Atlântica
BiomasBR | Julho 2025

Sumário

Carta ao Leitor

O presente guia foi elaborado com o objetivo de tornar o uso do pacote `sits` no R mais acessível para os profissionais que atuam no monitoramento da vegetação nativa no âmbito do projeto BiomasBR. Sabemos que há desafios no uso da linguagem R aplicada a rotinas operacionais de classificação, e, por vezes, isso cria uma barreira real para parte da equipe que está na linha de frente da análise ambiental e deseja tornar o processo de trabalho mais dinâmico.

Este material parte de uma premissa simples: oferecer os fundamentos essenciais para profissionais mais experientes que, por diversas razões, não tiveram contato prévio com linguagens de programação, mas que acumulam décadas de conhecimento prático, territorial e institucional.

A proposta deste documento é apresentar uma linguagem clara, com exemplos comentados passo a passo e explicações pensadas para quem nunca escreveu uma linha de código. O foco está no uso prático do `sits`, do ponto de vista de quem precisa entregar resultados, gerar mapas e interpretar classificações, mesmo sem pretensão de se tornar desenvolvedor ou programador.

Se você é alguém que sente dificuldade com termos técnicos ou acha que nunca vai conseguir usar o R, este guia é para você. E, se você já domina esses processos, pedimos que utilize este material para apoiar seus colegas, melhorando-o, compartilhando suas experiências e fortalecendo uma cultura de colaboração.

Aluizio Brito Maia

1 Introdução

Este documento tem como objetivo descomplicar o uso do pacote `sits` no R, voltado para aplicações práticas no contexto do BiomasBR. O foco está em fornecer um guia passo a passo acessível, com exemplos reais, boas práticas e resolução de erros comuns. O foco está nas atividades operacionais do BiomasBR, especialmente para colegas que nunca trabalharam com programação. O `sits` é um pacote da linguagem R desenvolvido para processar séries temporais de imagens de satélite, permitindo classificar automaticamente alvos ao longo do tempo. Mais detalhes podem ser adquiridos no livro do sits:

Rolf Simoes, Gilberto Camara, Gilberto Queiroz, Felipe Souza, Pedro R. Andrade, Lorena Santos, Alexandre Carvalho, and Karine Ferreira. Satellite Image Time Series Analysis for Big Earth Observation Data. Remote Sensing, 13, p. 2428, 2021. Disponível em: https://e-sensing.github.io/sitsbook/

Neste guia, você vai aprender a:

- Preparar os arquivos de imagem e organizar os dados;
- Criar um cubo de séries temporais.
- Inserir amostras de referência e treinar um modelo de classificação;
- Classificar grandes áreas automaticamente;
- Exportar os resultados para gerar mapas e análises.

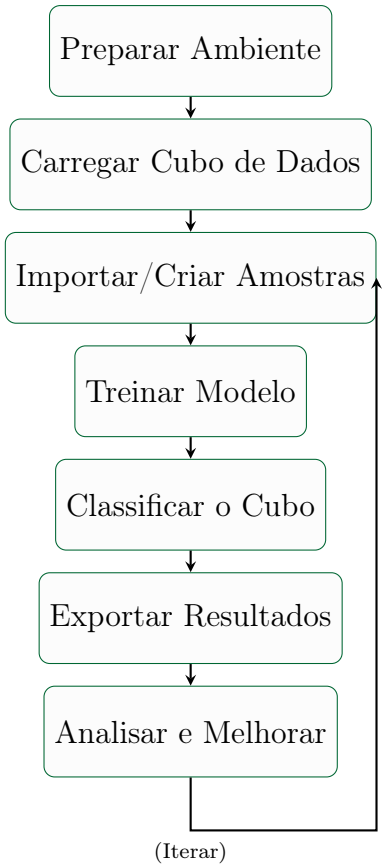
O guia está dividido em capítulos curtos e diretos, com exemplos comentados e códigos prontos para copiar e colar. Você pode seguir o passo a passo completo ou ir direto para a seção que precisa, como "Treinamento do Modelo" ou "Exportar o Mapa Final".

O objetivo é que o `sits` deixe de ser uma caixa-preta para se tornar uma ferramenta de trabalho compreensível e útil para todos e todas da equipe.

Qual é o nosso passo a passo? O que teremos ao final de todo esse processo?

Muitas vezes, somos ensinados a rodar códigos ou seguir tutoriais sem entender **o porquê de cada etapa**.

Aqui, vamos ver **de forma visual e simples** o caminho que percorremos com o `sits`, desde a preparação dos dados até a obtenção de um mapa classificado. Esse processo é **iterativo**, ou seja: pode ser repetido, melhorado e ajustado conforme ganhamos confiança e novos dados.



Descrição do Fluxo

Este fluxograma representa o processo de classificação de dados com o pacote `sits`.
Cada etapa é iterativa, permitindo ajustes conforme a qualidade dos resultados obtidos. A recomendação é reavaliar sempre após cada rodada de classificação.

Esse processo não é linear, é processual. Quanto mais você pratica, mais consegue ajustar e melhorar seus mapas.

2 Plantando o Primeiro Broto: Instalação e Ambiente

Para começar, é importante entender que existem dois programas diferentes que você vai usar juntos: **R** e **RStudio**.

- **R** é o programa principal: ele é o **motor** do nosso carro. É quem realmente faz os cálculos e executa os comandos.
- **RStudio** é a parte que a gente vê e usa: o **painel de controle**, o volante, o banco confortável. É nele que você vai trabalhar.

Você precisa instalar os dois, mas vai **usar sempre o RStudio** para seguir este guia.

Você pode baixar os dois nos seguintes sites oficiais:

- R: `https://cran.r-project.org/bin/windows/base/`
- RStudio: `https://posit.co/download/rstudio-desktop/`

Importante!

Se você estiver usando Windows, talvez em algum momento precise do RTools. O RTools é um conjunto de ferramentas que permite ao R compilar e instalar pacotes diretamente do código-fonte. Em outra palavras, pense no RTools como uma caixa de ferramentas extra que o R às vezes precisa para fazer certos trabalhos.

Você pode baixar o Rtools aqui:

- `https://cran.r-project.org/bin/windows/Rtools/rtools45/rtools.html`

Importante: instale o R primeiro, depois o `RStudio`, e só depois o `RTools`. E sempre abra o `RStudio` para usar o que será mostrado neste guia.

Resumindo: `R` é o que faz tudo funcionar, e o `RStudio` é o lugar onde você vai trabalhar. Mesmo que pareça estranho no começo, com o tempo você vai se acostumar!

E já que estamos no carrinho, vale mais uma analogia importante:

Os pacotes no R são como as **peças do carro**: o volante, o pedal de freio, os faróis, o banco, o limpador de para-brisa. Cada pacote adiciona uma função diferente ao carro. Sem eles, o motor liga, mas você não consegue dirigir direito.

Pacotes: como instalar e usar

Como vimos, os **pacotes** são as peças do nosso carro no R. E, pra usá-las, trabalhamos com duas funções importantes:

- `install.packages("nome")`: usamos essa função **apenas uma vez**, quando queremos instalar o pacote no nosso computador. É como comprar uma peça nova pro carro.
- `library(nome)`: usamos essa função **toda vez que quisermos usar o pacote em um projeto**. É como encaixar a peça no carro e começar a dirigir.

Por exemplo, para instalar o pacote `sits`, usamos:

```
install.packages("sits")
```

E sempre que quisermos usar o `sits`, por exemplo, dentro do RStudio, usamos:

```
library(sits)
```

Importante: instalar é uma vez só em! Mas carregar com `library()` é sempre que for usar.

Neste guia, vamos usar alguns pacotes que "instalam" as funções que precisamos para fazer o trabalho acontecer.

Os pacotes essenciais são:

```
install.packages("sits")
install.packages("dplyr")
install.packages("sf")
library(sits)
library(dplyr)
library(sf)
```

Conforme forem sendo usados novos pacotes, vamos os instalando.

3 Abrindo a Clareira: Criando o Cubo

O primeiro passo é criar um cubo, que será usado como base para todas as análises. Usamos então o código abaixo:

```
library(sits)

meu.cubo.bonito <- sits_cube(
  source = "BDC",
  collection = "SENTINEL-2-16D",
  tiles = Coloque aqui seu tile,
  bands = c("NDVI", "EVI"),
  start_date = "2019-09-30",
  end_date = "2020-09-29"
)
```

Explicações de cada item:

- meu.cubo.bonito: é o nome da sua variável. Você pode dar o nome que quiser, desde que siga as regras do R. Sempre que você usa <-, está dizendo: "esse nome vai guardar esse conteúdo".
- o sitscube guarda a função, toda função deve ser aberta com "("e fechada com ")", não se esqueça!! Essa função que vai conter os itens abaixo:
- `source`: indica a origem dos dados. No nosso caso, usamos `"BDC"` (Brasil Data Cube).
- `collection`: aqui você escolhe qual satélite/sensor será usado. Para saber quais coleções estão disponíveis, use:

  ```
  sits_list_collections()
  ```

- `tiles`: refere-se às regiões geográficas (tiles) que você vai usar. Essa parte precisa ser preenchida com o código do tile que deseja processar (ex: `"033027"`).
- `bands`: define quais bandas ou índices serão utilizados (ex: NDVI, EVI, NDWI).

  Substitua "Sentinel-2-16D"pela coleção que for usar, se for o caso de usar qualquer outra.

4 Coletando Sementes: Criando ou Importando Amostras

As amostras são os primeiros sinais de vida: com elas, ensinamos o modelo a reconhecer cada pedaço da paisagem.

No capítulo anterior, criamos o nosso cubo de dados, uma espécie de mapa temporal com imagens Agora, vamos aprender como criamos ou importamos amostras para o ambiente do `sits`.

Afinal, o que são amostras?

Amostras são pequenos pontos, áreas ou regiões que a gente já conhece. Sabemos que ali tem floresta, ou pasto, ou água, por exemplo. Essas informações ajudam o R a "aprender" a reconhecer esses padrões no resto da imagem.

Em outras palavras, coletamos pontos e interpretamos o pertencimento daquelas classes, quase como o trabalho de interpretação já realizado no `PRODES`. A partir disso, rotulamos essas amostras, ou seja: **damos nomes aos bois**.

Coletando amostras no TerraCollect.

Importando Amostras via Shapefile

Tendo as nossas amostras coletadas (no próprio R ou em ferramentas como o TerraCollect/QGIS), podemos importá-las para o ambiente do R. Para isso, usamos o código abaixo:

```
amostras <- sf::st_read("caminho//das//amostras.shp")
```

Entendendo o código:

- `amostras` é o nome da nossa variável. você pode escolher o nome que quiser.
- `sf` é a biblioteca usada para trabalhar com dados espaciais vetoriais. Ela deve ser carregada antes com `library(sf)`.

- `st_read()` é a função da biblioteca `sf` usada para ler arquivos shapefile (`.shp`), geopackage (`.gpkg`), entre outros.
- O caminho entre aspas deve apontar corretamente para onde suas amostras estão salvas no computador.

Boas práticas na coleta de amostras

- Cada classe deve ter um número razoável de amostras, evite desequilíbrios extremos.
- Evite amostras em áreas confusas (bordas de classes, nuvens, sombras, transições).
- Dê nomes simples, claros e consistentes às suas classes (ex: `"Floresta"`, `"Pasto"`, `"Solo Exposto"`).

E lembre-se:

Assim como uma floresta começa com sementes, uma boa classificação começa com boas amostras.

5 Ensinando a Floresta a Falar: Treinando o Modelo

Agora que já temos nosso cubo (o mapa temporal das paisagens) e as nossas sementes (as amostras que coletamos), vamos treinar um algoritmo de aprendizado de máquina, no nosso caso, o escolhido é o Random Forest. Pense nele como um analista incansável: você ensina o que é floresta, água, desmatamento, e ele aplica esse aprendizado para classificar toda a imagem, de forma automática e rapidíssima. Ele busca padrões nos dados, compara com o que você mostrou e preenche o mapa como se estivesse vetorizando tudo sozinho.

Pra isso, usamos o seguinte código:

```
set.seed(025035)

modelo_rf <- sits_train(
    samples = amostras,
    ml_method = sits_rfor()
)
```

Entendendo o código:

- `set.seed(025035)`: Esse comando define uma "semente" de aleatoriedade. Em outras palavras: ele garante que, mesmo sendo um processo com variações, o seu resultado será sempre o mesmo toda vez que rodar esse código. Isso é importante para garantir reprodutibilidade, ou seja, para que outras pessoas consigam obter o mesmo resultado. Você pode usar qualquer número aqui, desde que mantenha o mesmo se quiser repetir o experimento depois.
- `modelo_rf`: É o nome que estamos dando para o nosso modelo treinado. Pode ser o nome que você quiser (desde que siga as regras do R),
- `samples = amostras`: Aqui é onde dizemos para o algoritmo quais são as nossas sementes, ou seja, as amostras rotuladas que ele vai utilizar para aprender.
- `ml_method = sits_rfor()` : É o coração do comando: define qual método de aprendizado de máquina vamos usar. No caso, o `sits_rfor()` é o atalho para dizer "quero usar o algoritmo Random Forest".

Depois de treinar o nosso modelo com o algoritmo Random Forest, podemos perguntar pra ele: "Ei, modelo... o que mais te ajudou a tomar suas decisões?"

É isso que fazemos ao usar o comando:

```
plot(modelo_rf)
```

Ele te volta uma informação assim:

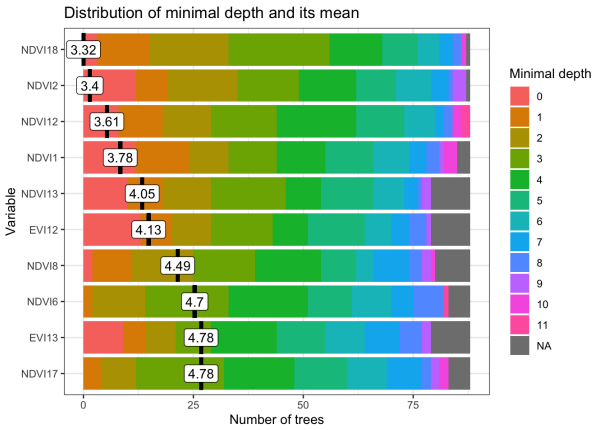


Figura 5.1: Importância das Variáveis.

Cada barra representa uma variável (como NDVI ou EVI em diferentes datas). As cores mostram em que nível da "árvore de decisão" essa variável apareceu. Variáveis que aparecem com cores mais quentes e barras maiores (como a NDVI18) são as que o modelo mais confiou para separar as classes.

Ou seja: esse gráfico nos dá uma ideia do que o modelo "achou mais importante" ao classificar o ambiente.

E o que eu faço com isso?

Esse gráfico serve como um mapa de pistas para você entender o que o modelo considerou mais importante na hora de classificar a paisagem.

Você pode usar essa informação para:

Avaliar a qualidade do seu cubo: Se uma única variável está dominando tudo, pode ser sinal de que as outras não estão trazendo muita informação nova.

Eliminar variáveis pouco úteis: Se alguma banda aparece sempre no fundo da árvore (com cor fria e profundidade alta), talvez ela não esteja ajudando tanto, e pode ser deixada de fora em futuras análises.

Entender o comportamento das classes: Às vezes, uma banda que você achava irrelevante é justamente a que mais ajuda a separar desmatamento de vegetação, por exemplo.

6 Mapeando os Caminhos da Mata: Classificando os Dados

Depois de treinar nosso modelo, chegou a hora de usá-lo: vamos aplicá-lo sobre o cubo e gerar nosso primeiro mapa classificado.

Mas antes... qual é a diferença entre o **modelo** e o **mapa**?

Pense no modelo como um *guia experiente da floresta*. Ele foi treinado com exemplos e sabe identificar onde tem água, onde tem floresta, onde há pasto.

Já o mapa classificado é o *registro final*, onde ele aponta e diz: "aqui é floresta", "aqui é solo exposto".

Em outras palavras:

- O **modelo** é o cérebro, o conhecimento aprendido.
- O **mapa classificado** é a aplicação prática desse conhecimento sobre todo o território do cubo.

É como ensinar alguém a reconhecer pegadas de animais. Depois de aprender, essa pessoa percorre toda a trilha e faz um **mapa das pegadas encontradas**: e esse é o nosso produto final.

Mas nem tudo é tão definido: Probabilidade antes da certeza

Quando usamos o modelo para classificar o cubo, a primeira coisa que obtemos **não é o mapa final**, mas sim um **mapa de probabilidades**.

Cada pixel do cubo recebe uma pontuação:

> "Com `X%` de chance, isso aqui é floresta. Com `Y%`, é água. Com `Z%`, pode ser pasto..."

Ou seja: é um mapa *fuzzy*, onde a classificação ainda está "meio indecisa".

Para gerar esse mapa de probabilidades, usamos o seguinte código:

```
mapa_prob <- sits_classify(
    data = meu.cubo.bonito,
    ml_model = modelo_rf,
```

```
    multicores = 2,
    memsize = 8,
    output_dir = "caminho/para/salvar"
)
```

Esse mapa é muito útil para entender onde o modelo está **confiante** e onde ele está **inseguro**.

E só depois vem o mapa final: a partir do mapa de probabilidades, escolhemos a classe com maior chance para cada pixel. Isso é o que chamamos de **decisão hard**, ou seja, transformar o "talvez floresta" em "isso *é* floresta", mesmo que a certeza não seja de 100%.

Esse é o nosso **mapa classificado final** e pode ser gerado por:

```
mapa_final <- sits_label_classification(
    output_dir = "caminho/para/salvar"
)
```

Visualizamos ele com:

```
plot(lem_map)
```

Corrigir primeiro o erro, depois a incerteza

Um bom passo seguinte é analisar onde o modelo erra com mais frequência. E aí vem uma dica valiosa:

> **Comece pelos lugares onde a probabilidade foi baixa**, ou seja, onde o modelo ficou inseguro.

Esses pontos são ótimos candidatos para revisão: talvez a amostra esteja mal rotulada, talvez o dado esteja ruim, talvez a classe seja ambígua. O `mapa de probabilidades` te guia como uma *lanterna*, iluminando onde vale a pena revisar com mais cuidado.

Depois que os **erros mais evidentes** forem tratados, você pode ir refinando as *zonas de incerteza*, áreas que estão no limiar entre duas classes, por exemplo.

Dica extra: Visualização

Você pode visualizar o mapa de probabilidades com o seguinte comando:

```
plot(lem_probs, labels = "Floresta", palette = "Greens")
```

Isso ajuda a ver com os próprios olhos onde a floresta está **bem definida (probabilidades altas)** e onde o modelo ficou mais em dúvida.

Boas Práticas

- Use sempre o mapa de probabilidades antes de gerar o mapa final.
- Dê atenção especial às áreas com baixa confiança.
- Revise amostras e tiles problemáticos com base nas zonas incertas.
- Documente suas decisões: mudanças feitas, causas encontradas e como elas afetam o modelo.

7 Espalhando as Folhas: Exportando e Visualizando Resultados

Depois de classificar nosso cubo, é hora de transformar esse conhecimento em algo visível e compartilhável: os **mapas finais**. É como se agora, depois de plantar, crescer e organizar a floresta, a gente estivesse espalhando suas folhas.

O pacote `sits` nos permite exportar os resultados de forma simples, gerando arquivos GeoTIFF que podem ser abertos em qualquer SIG (como QGIS, TerraAmazon, etc).

Para exportar o mapa classificado, usamos:

```
    sits_export(
    data = mapa_prob,
    filename = "meu_mapa_classificado.tif"
)
```

Esse comando salva o resultado do processo de classificação no seu computador. O arquivo gerado pode ser usado em análises espaciais, cruzado com outras camadas, ou até mesmo impresso como mapa para relatórios técnicos.

Visualizando dentro do R

Se quiser visualizar o resultado ainda dentro do ambiente R, também é possível! Por exemplo, para ver a distribuição da classe "Floresta" no mapa de probabilidades:

```
plot(lem_probs, labels = "Floresta", palette = "Greens")
```

E para visualizar o mapa final (após a decisão hard), você pode fazer:

```
mapa_prob <- sits_classify(
    data = meu.cubo.bonito,
    ml_model = modelo_rf,
    output_dir = "caminho/para/salvar",
    output_type = "labels"
)

plot(lem_mapa_final)
```

Isso mostra, em cores distintas, a distribuição das classes no seu território.

Dica Final

Boas Práticas

- Sempre nomeie seus arquivos de forma clara: inclua o ano, o tile, ou o tipo de dado (ex: `classificacao_NDVI_tile33W.tif`).
- Mantenha um backup dos resultados exportados — eles podem ser usados em análises futuras ou reprocessamentos.

8 Desatando os Cipós: Dicas, Erros e Boas Práticas

Agradecimentos

Aos colegas do BiomasBR, pelo apoio constante, e à comunidade que desenvolve o `sits` com excelência. Que este guia ajude a tornar o processamento de séries temporais mais acessível, transparente e confiável.