Entregável 2.2. Relatório com resultados comparativos PyCCD vs PyCCD+

Índice

Índice das Tabelas

Índice das Figuras

1 Introdução

Para distinguir o treino do algoritmo e a criação de cartas de perdas de vegetação, é fundamental compreender o propósito e os processos envolvidos em cada uma destas etapas. Enquanto o treino do algoritmo foca-se na escolha e otimização dos parâmetros para assegurar um modelo robusto e preciso, a criação de cartas de perdas de vegetação aplica este modelo para gerar visualizações úteis e informativas das áreas afetadas. Cada etapa tem a sua importância específica e, juntas, formam um processo completo para a monitorização e gestão das perdas de vegetação.

- **Treino do Algoritmo**

O treino de um algoritmo é uma fase crucial no desenvolvimento de modelos preditivos e analíticos. Este processo envolve a escolha e ajuste dos melhores valores para os parâmetros do algoritmo, garantindo que ele aprenda a partir dos dados de treino e possa fazer previsões precisas e robustas. A escolha dos parâmetros é uma tarefa delicada que requer experimentação e validação contínua (ref. Didática Tech. - Dados de Treino e Teste).

Após a exploração inicial dos dados e definição dos parâmetros iniciais do algoritmo, é utilizada uma base de dados de referência (no nosso caso utilizámos a BDR-300_DGT) para avaliar o desempenho do modelo. São utilizadas métricas como o F1-Score, Erro de Omissão e Erro de Comissão para ajudar a determinar se o modelo está bem calibrado e se os erros associados são mínimos.

- **Criação de Cartas de Perdas de Vegetação**

A criação de cartas de perdas de vegetação é uma aplicação prática que irá utilizar o modelo treinado para mapear e quantificar áreas de vegetação perdida ao longo do tempo. Este processo pode ser menos dependente de uma validação contínua, focando-se mais na aplicação do modelo desenvolvido.

Utilizam-se imagens Sentinel-2 (Theia/Maja ou GEE/S2cloudness) e o modelo treinado é aplicado para prever as áreas de perda de vegetação. As previsões são convertidas em mapas geográficos (ou cartas) que visualizam a extensão e localização das perdas.

Para implementar eficientemente o CCDC, foi feita uma adaptação de um código Python disponível em https://code.usgs.gov/lcmap/pyccd, originalmente projetado para dados do satélite LANDSAT. No âmbito deste projeto, o código foi adaptado para suportar dados do satélite Sentinel-2, resultando numa versão otimizada e mais rápida do PyCCD, que está disponível em: https://github.com/manuelcampagnolo/S2CHANGE.

Neste relatório, são descritas as potenciais adaptações tecnológicas para implementação na cadeia de produção da DGT, seguindo a seguinte ordem:

1. Dados: esta seção aborda os diferentes tipos de dados que foram utilizados e testados no algoritmo PyCCD.
2. Parâmetros atuais do PyCCD: são discutidos os parâmetros específicos do modelo PyCCD que foram ajustados para otimizar o seu desempenho.
3. Pré-processamento dos dados: explicação do pré-processamento que é feito aos dados antes de iniciar o algoritmo.
4. Processamento do PyCCD e criação de carta de perdas de vegetação: explicação da aplicação do algoritmo aos dados pré-processados. A principal saída do algoritmo irá ser a criação de uma carta de perdas de vegetação.

5. Componentes do algoritmo com maior peso no processamento: identificação das partes do algoritmo que consomem o maior tempo de processamento.
6. Estratégias para reduzir o tempo de computação: são apresentadas estratégias para diminuir o tempo de computação, incluindo melhorias na leitura dos dados e a otimização do algoritmo, como a aceleração do treinamento da regressão LASSO.

2 Dados

Para testar e validar o algoritmo, utilizou-se dados das coleções de imagens de satélite Theia/MAJA e GEE/S2cloudless. Eventualmente mais tarde poderá ser testado dados do Copernicus/Sen2cor.

O treino e a parametrização do modelo são fases cruciais para garantir a precisão e a confiabilidade do nosso algoritmo. Recorreu-se à Base de Dados de Referência DGT-300 para validar e treinar o nosso algoritmo, sendo essencial para assegurar a precisão e a confiabilidade do nosso modelo. Numa fase posterior do projeto, será também aplicado o PyCCD a uma nova Base de Dados de Referência — Navigator.

3 Parâmetros atuais do algoritmo PyCCD

O algoritmo PyCCD está estruturado em duas etapas principais: o pré-processamento dos dados e a aplicação do PyCCD. Os parâmetros atuais do algoritmo foram ajustados para otimizar o desempenho do modelo, sendo os seguintes:

- 'PEEK_SIZE': 6
- 'MIN_YEARS': 1
- 'DETECTION_BANDS': NDVI, Green, SWIR2
- 'TMASK_BANDS': Green, SWIR2
- 'LASSO_MAX_ITER': 1000
- 'ALPHA': 20
- 'CHISQUAREPROB': 0.999

Estes parâmetros foram ajustados com base em análises e validações anteriores (**Entregável 1.2.**), estabelecendo diferenças significativas em relação aos parâmetros do Moraes et al., 2024, que utilizaram o CCD do GEE (eCCD), com **'ALPHA'** e **'LASSO_MAX_ITER'** definidos como 200 e 25000, respetivamente. Ao testar o nosso algoritmo PyCCD com 10.000 pontos utilizando os parâmetros acima mencionados, constatamos um desempenho superior ao modelo CCD do GEE, quando ambos são avaliados com a mesma Base de Dados de Referência. Isto é evidenciado pelas métricas de F1-Score e Erro de Comissão detalhadas na Tabela 1.

	F1-Score [%]	Erro de Omissão [%]	Erro de Comissão [%]
PyCCD THEIA/MAJA	82.34	15.90	18.38
eCCD GEE/s2cloudless	81.14	15.77	21.73

Tabela 1 – Desempenho do algoritmo PyCCD com dados THEIA/MAJA e eCCD com dados GEE/s2cloudless.

4 Pré-processamento e leitura dos dados

Para otimizar a leitura da série temporal de cada pixel, a coleção de imagens Sentinel-2 passa por um processo de pré-processamento (Figura 1).

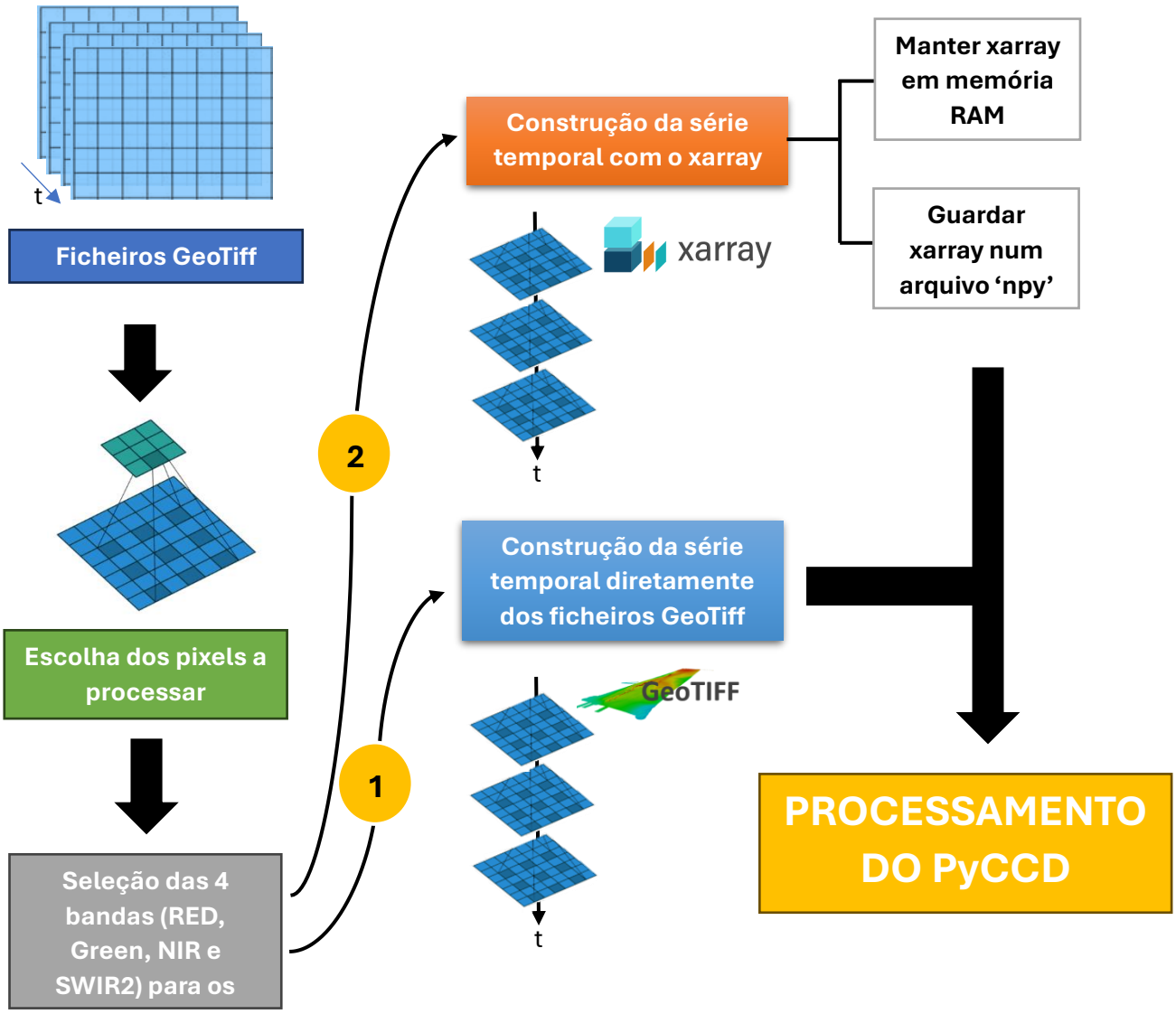

Figura 1 – Etapas do pré-processamento aos dados: (1) método a seguir por GeoTiff e (2) método a seguir por xarray.

- **Seleção dos pixels**

Os valores dos pixels relevantes são inicialmente extraídos dos arquivos GeoTiff para serem processados pelo PyCCD. É essencial selecionar cuidadosamente os pixels para economizar tempo de processamento, pois o algoritmo não é aplicado a todos, como por exemplo áreas edificadas (cf. discutido no **Entregável 2.1.**). A delimitação destas áreas envolve o processamento intensivo de dados, abrangendo aproximadamente 120 milhões de pixels. Em relação à magnitude dos dados envolvidos, o projeto enfrenta o desafio de

lidar com cerca de 0.8 Tb de informações, correspondendo a uma estimativa computacional de aproximadamente 34.560 *core-hours* para processar uma série temporal de 8 anos. Estes dados incluem arquivos '*npy*' que armazenam séries temporais de 120 milhões de pixels ao longo do período mencionado.

- **Seleção das bandas específicas**

Além do recorte dos pixels, também ocorre uma seleção das bandas específicas necessárias para nosso modelo: RED, GREEN, NIR e SWIR2. Esta escolha adicional ajuda a reduzir ainda mais o tamanho dos arquivos, focando-se apenas nas informações essenciais para a análise das mudanças na vegetação ao longo do tempo. Especificamente, as bandas RED e NIR são selecionadas para calcular o Índice de Vegetação por Diferença Normalizada (NDVI), que é posteriormente utilizado como uma das variáveis no algoritmo. A escolha de usar apenas as bandas NDVI, GREEN e SWIR2 no processamento do PyCCD deve-se às propriedades espectrais destas bandas, que são mais sensíveis às mudanças na vegetação.

- **Opções de pré-processamento**

Existem duas opções de pré-processamento para preparar os dados antes de aplicar o PyCCD:

1. Ler a série temporal diretamente dos ficheiros GeoTiff, os testes realizados mostraram que esta opção pode levar ~ 3 horas para processar 1 km^2, sendo a forma menos eficiente em termos de tempo de computação.
2. Construir a série temporal das bandas para todos os pixels a serem processados através do xarray. Nesta abordagem, há duas opções:
    - Quando a matriz resultante do xarray é mantida na memória RAM pelo programa, isso implica que os dados precisam ser lidos novamente sempre que o PyCCD for processado para os mesmos pixels. Esta escolha pode resultar num aumento significativo do tempo de processamento do PyCCD, conforme evidenciado pelos testes que demonstraram que esta abordagem levou três vezes mais tempo em comparação com a opção de salvar os dados num arquivo '*npy*' no disco.
    - A matriz do xarray é armazenada no disco no formato .npy, com as dimensões: **número de datas x número de bandas x número de pixels**. Cada linha desta matriz representa uma data que contém as 4 bandas para todos os pixels. Isto permite ignorar o tempo de leitura durante o processamento do PyCCD, possibilitando, por exemplo, que uma única máquina realize este pré-processamento e crie os arquivos '*npy*'. Esta abordagem foi testada e demonstrou o menor tempo de processamento em comparação com outras opções avaliadas, tendo demorado ~ 28 minutos a processar 1 km^2.

Manter os dados no disco facilita o acesso quase instantâneo aos arquivos '*npy*', consumindo muito pouco tempo para serem acedidos. Além disso, há um código Python disponível no nosso repositório aberto do GitHub (https://github.com/manuelcampagnolo/S2CHANGE/blob/main/scripts/pyccd_theia/notebooks/addNewImageToFile.py) que permite a adição de novas imagens da coleção

Sentinel-2 a estes arquivos '*npy*' de forma eficiente, facilitando a incorporação de novos dados à matriz rapidamente. Esta adição ocorre instantaneamente, e encontra-se detalhada no esquema seguinte:

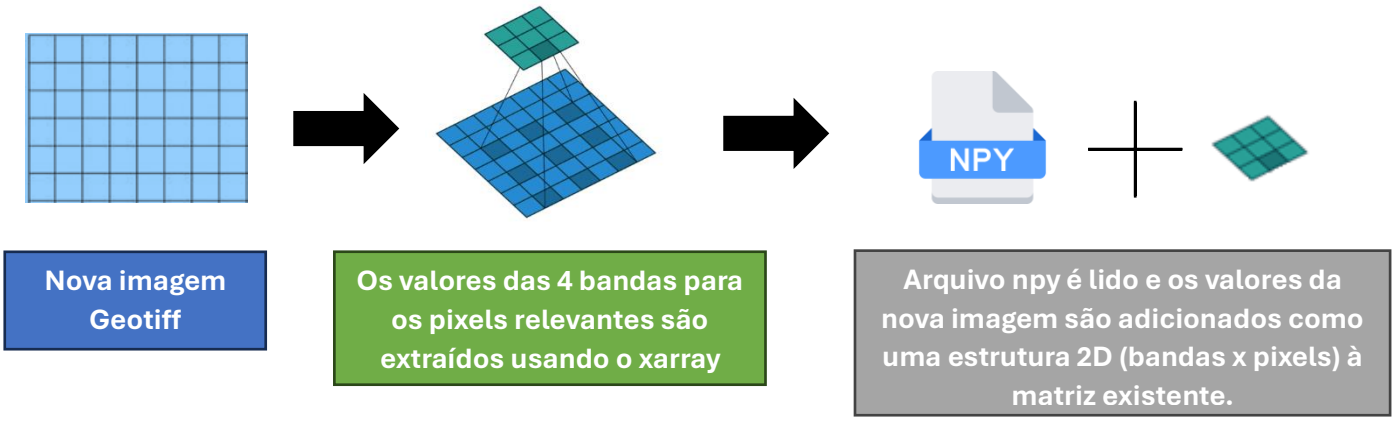

Figura 2 – Processo de adição de uma nova imagem Geotiff ao arquivo npy existente.

5 Processamento e criação de carta de perdas de vegetação

Na segunda etapa, o PyCCD é aplicado a cada pixel previamente processado. Este algoritmo identifica mudanças significativas na vegetação através da análise de séries temporais e da deteção de quebras, ajustando curvas de regressão LASSO.

Durante esta etapa, as bandas são filtradas para eliminar os dias em que não há dados disponíveis (*nodata*). O resultado final do algoritmo consiste na extração de várias informações para cada pixel processado, incluindo uma lista de datas com coeficientes harmônicos associados a cada intervalo entre elas, a magnitude das mudanças no NDVI e a localização geográfica do pixel. Estas informações irão ser úteis para a produção da carta de perdas de vegetação.

5.1 Componentes do algoritmo com maior peso no processamento

Os ajustes realizados por estas duas regressões consomem a maior parte do tempo de processamento do algoritmo, destacadas a salmão na Figura 3:

1. **Ajuste robusto (*Robust fit*):** é utilizado para lidar com '*outliers*' nos dados, consumindo aproximadamente 20% do tempo total do processamento do algoritmo. Esta função assegura que observações atípicas não influenciem excessivamente os resultados do algoritmo.
2. **Regressão LASSO:** é responsável por cerca de 77% do tempo de processamento. A regressão LASSO é ajustada para cada nova observação na série temporal, o que representa um consumo significativo de tempo de processamento.

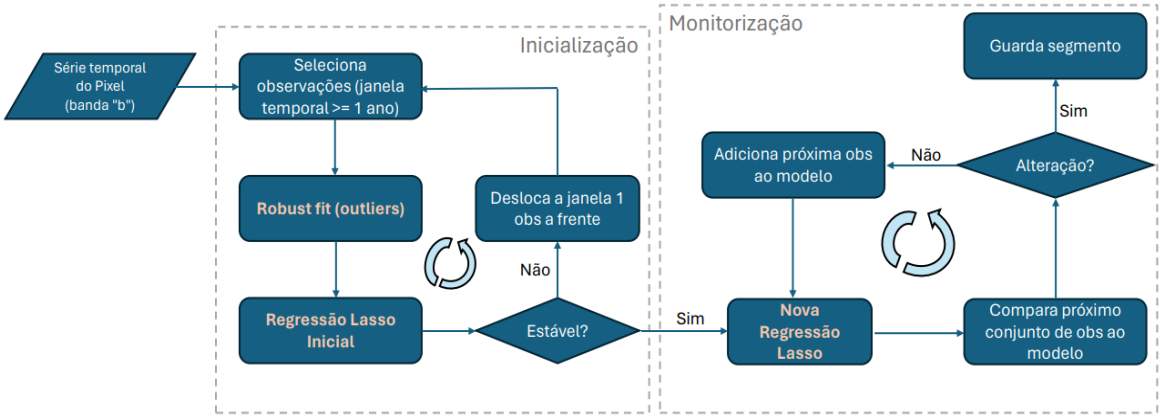

Figura 3 – Esquema simplificado do funcionamento do algoritmo PyCCD.

Além destes fatores, a complexidade das operações matemáticas envolvidas nestas regressões e a necessidade de processamento intensivo também contribuem para o tempo total necessário para executar o CCD de forma eficiente.

5.2 Estratégias a explorar para reduzir o tempo de computação

Existem algumas estratégias ainda a explorar para reduzir o tempo de computação do PyCCD, nomeadamente:

1. Adaptação do algoritmo PyCCD**:** Otimização do algoritmo ao reduzir o número de chamadas à função LASSO e agrupar observações em vez de processá-las individualmente. Além disso, foi explorada a possibilidade de implementar a regressão LASSO em Cython, uma linguagem que combina Python com C, para obter ganhos significativos de desempenho computacional.
2. Identificação de padrões com machine learning: Utilizar técnicas de classificador para identificar pixels com trajetórias semelhantes. Processar o PyCCD apenas para um pixel representativo de cada grupo, transmitindo as informações relevantes do pixel representativo (data de quebra, magnitude, etc.). Esta abordagem ainda requer testes.
3. Reaproveitar os segmentos já calculados (como mencionado no **Entregável 2.1.**): processar o PyCCD apenas a partir da última data de quebra. Evita processar o algoritmo desde o princípio sempre que se pretenda estender a série temporal.
4. Utilização de recursos de HPC**:** Explorar o uso de serviços computacionais de High Performance Computing (HPC), como a RNCA, INCD, para estimar o tempo de execução do CCD em supercomputadores.

6 Conclusões

Ao concluir este relatório, identificámos as seguintes potenciais adaptações tecnológicas a serem implementadas na cadeia de produção da DGT:

- Implementar uma etapa de pré-processamento aos dados e à sua leitura é altamente recomendável para otimizar o fluxo de trabalho do PyCCD.

- Durante o processamento do PyCCD, é crucial considerar as especificações das máquinas utilizadas, como o número de núcleos e a velocidade do processador, para maximizar a eficiência computacional.
- É essencial desenvolver uma versão aprimorada do algoritmo para garantir eficiência operacional nas máquinas da DGT.

7 Referências

1. Código PyCCD para dados LANDSAT: https://code.usgs.gov/lcmap/pyccd (Acedido em janeiro 2024).
2. Didática Tech. Dados de Treino e Teste. Disponível em: https://didatica.tech/dados-de-treino-e-teste/ (Acedido a: 21 de junho 2024).
3. Entregável 2.1.: https://github.com/manuelcampagnolo/S2CHANGE/blob/main/Entregavel_1_2_v2.pdf
4. Moraes, D., Barbosa, B., Costa, H., Moreira, F. D., Benevides, P., Caetano, M., & Campagnolo, M. (2024). Continuous forest loss monitoring in a dynamic landscape of Central Portugal with Sentinel-2 data. International Journal of Applied Earth Observation and Geoinformation, 130, 1-12. Article 103913. Advance online publication. https://doi.org/10.1016/j.jag.2024.103913
5. Repositório GitHub: https://github.com/manuelcampagnolo/S2CHANGE