Ruprecht-Karls-Universität Heidelberg

GEOGRAPHISCHES INSTITUT HEIDELBERG

Computer Science for Geographers

Sommersemester 2019

Tiled NDVI Analysis

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Dozentin:

Christina Ludwig

Vorgelegt am 31.10.2019

von

Julian Vetter (Matr.-Nr: 3547523)

Inhaltsverzeichnis

1. Einleitung

Ein großer Anteil von wissenschaftlichen Arbeiten greift in ihren Analysen auf Software-Tools zurück um diverse Modellierungen oder Berechnungen durchzuführen. Oftmals werden diese Tools aufgrund der spezifischen Anforderungen von den Wissenschaftlern selbst geschrieben. Hierbei entsteht jedoch das Problem, dass Wissenschaftlern zumeist das Wissen über *Best Practice*-Verfahren aus dem Bereich der Softwareentwicklung fehlt (Wilson et al. 2014, Wilson et al. 2017). Dies kann dazu führen, dass ungewollt unzuverlässiger und unübersichtlicher Code produziert wird, der anfällig für Errors und für andere oftmals nicht nachvollziehbar ist (ebd.). Um dies zu verhindern, haben Wilson et al. (2014 & 2017) eine Reihe von *Best*- sowie *Good-Enough-Practices* aufgestellt. Im Folgenden wird darüber reflektiert, welche von diesen Praktiken für die Abschlussarbeit im Rahmen des Seminars *Computerscience for Geographers* angewendet wurden. Für die Abschlussarbeit sollte ein Programm geschrieben werden, welches die Differenz des NDVI's (*Normalized Difference Vegetation Index*) von zwei unterschiedlichen Zeitpunkten berechnet. Da die Bearbeitung von größeren Satellitenbildern oftmals sehr RAM-intensiv sein kann, sollte die Berechnung des NDVI's gestückelt durchgeführt werden. Weiterhin wurde die Verwendung eines API's (*Application Programming Interface*) zur Automatisierung der Satellitenbildsuche gefordert.
Der Aufbau der folgenden Kapitel orientiert sich an Wilson et al. (2014). Es wird grundsätzlich dargelegt, welche der *Best Practices* wie im Code implementiert wurden.

2. Write programs for people, not computers

Um den Code verständlicher und leserlicher zu gestalten, wurde das Programm in Funktionen aufgeteilt, die jeweils eine spezielle Aufgabe übernehmen. Weiterhin konnten so unnötige Wiederholungen im Code vermieden werden. Den Funktionen wurden sinnvolle Namen im *pot_hole_case_style* zugewiesen. Diese Art Namensvergabe wurde auch für alle Variablen und Parameter adaptiert. Ebenfalls wurden keiner Funktion mehr als sechs Parameter zugewiesen. Somit soll gewährleistet sein, dass der Code einfach zu verstehen und nachzuvollziehen ist. Hierbei ist anzumerken, dass nach *Pylint*-Standards maximal fünf Parameter an eine Funktion übergeben werden sollen, nach Wilson et al. (2017) ist es jedoch legitim bis zu sechs Parameter pro Funktion zu verwenden.
Jede der Funktionen enthält einen *Docstring*, indem kurz erläutert wird welche Aufgabe die Funktion erfüllt und welche Parameter der Funktion übergeben werden müssen. Falls vorhanden wird ebenfalls auf den Return-Wert der Funktion eingegangen.

Um einzelne Code-Blocks durch Einrückungen voneinander zu trennen, wurde kontinuierlich die Tabulator-Taste verwendet. Der folgende Header der Funktion *search_image*, dient der Verdeutlichung der zuvor aufgeführten Aspekte.

```
def search_image(date, bounding_box, prop):
    """Searches Satellite-Image for given Boundingbox, Date and Properties
    :parameter:
    singel Date as string,
    Bounding Box as List
    Properties as String
    :return:
    statsac.Item Object with the lowest
    cloud-coverage for the given Date and bounding box"""
```

Abbildung 1: Header der Funktion search_image. Quelle: Eigene Darstellung

2. Let the computer do the work.

Das Programm wurde so strukturiert, dass der Nutzer zur Berechnung der NDVI-Differenz die Eingabeaufforderung nutzen kann. Falls eine größere Anzahl an Berechnungen durchgeführt werden muss, wird somit eine bequeme und einfache Schnittstelle zwischen Nutzer und Skript geboten.

3. Make Incremental Changes

Als VSC (*Version Control System*) wurde GIT verwendet. Wenn Veränderungen am Code vorgenommen wurden, wurde diese über einen Commit festgehalten und anschließend auf das Repository gepusht. Dies hat den Vorteil, dass so automatisch ein Versionsverlauf generiert wurde und bei eventuellen Fehlern im Code auf vorherige Versionen zurückgegriffen werden konnte. Die Entwicklung des Codes gliederte sich grundsätzlich wie folgt.
Zuerst wurden die Funktionen für die Satellitenbildsuche geschrieben. Diese greifen auf die SAT-Search-API zurück. Anschließend musste ein Filter implementiert werden, da die SAT-Search-API fehlerhafte Daten für Sentinel-Bilder liefert und somit nur Landsat-Bilder für das Skript verwendet werden können. Daraufhin wurde die Funktion für die Berechnung des NDVI's eingefügt und getestet, ob das *Broadcasting* und die Nulldivision funktionieren. Danach wurde eine Funktion implementiert, welche die Differenz zweier *Numpy*-Arrays berechnet. Diese wird wie auch die NDVI-Funktion von den *Tiling*- Algorithmen genutzt, damit die Verrechnung der *Tiles* sequentiell ohne redundante Zwischenschritte erfolgen kann. Im Anschluss wurde zuerst eine Funktion geschrieben, die für das Erstellen der Bildblöcke auf die interne Blockstruktur der Landsat-Bilder zurückgreift. Schließlich wurde ein Algorithmus

implementiert, der die Satelliten-Bilder in Benutzerdefinierte Blöcke einteilt und die entsprechenden Berechnungen mit den einzelnen *Tiles* durchführt.
Im Rahmen des Entwicklungsprozesses wurden diese Funktionen immer wieder vereinfacht und optimiert, sodass die aktuelle Code-Struktur entstand.

4. Don't Repeat Yourself (or Others)

Wie schon in Kapitel 2 vermerkt, wurde der Code in einzelne Funktionen gegliedert um unteranderem auch Wiederholungen zu dezimieren. Falls ein Error im Code auftreten sollte, muss dieser so nur in der entsprechenden Funktion und nicht an jeder Stelle, an der die Funktion genutzt wird, behoben werden. Weiterhin bietet dies den Vorteil, dass der Code hierdurch kürzer und damit auch übersichtlicher wird.

5. Plan for Mistakes

Für die Erhöhung der Zuverlässigkeit wurden für einige essentielle Funktionen mithilfe der *Nose*-Bibliothek Tests geschrieben. Unteranderem wurde die NDVI-Berechnung dahingehend getestet, ob die Zero-Division ist erlaubt und ob das Broadcasting der *Numpy*-Arrays funktioniert. Zusätzlich wurde überprüft, ob die erstellten *Custom-Tiles* auch tatsächlich die gewünschten Ausmaße in Metern besitzen. Weiterhin war es notwendig zu überprüfen, ob die *Custom* und *Optimal-Tiled*-Berechnungen auch dann funktionieren, wenn die angegebenen Raster-Bilder unterschiedlich groß sind bzw. verschiedene Shapes besitzen. Für diese Tests wurden abweichend vom gängigen Standard keine eigenen Datensets generiert, sondern auf bestehende Landsat-Raster-Bilder zurückgegriffen. Dies hat den einfachen Grund, dass die Funktionen nur bereits mit *Rasterio* geöffnete Datensätze oder *Statsac-Items* von der *Sat-Search* API als Parameter akzeptieren. Somit erweist es sich als recht komplex geeignete Datensätze speziell für die Tests zu generieren. Deshalb kann der Gebrauch von „vorgefertigten Datensätzen" mit unterschiedlichen Shapes zu Testzwecken als „*Good-Enough"* gerechtfertigt werden.
Neben den Automatisierten Tests wurden im Code selbst *Assert*-Statements verwendet um eventuelle Fehler abzufangen. Beispielsweise wird hiermit überprüft, ob die Suchanfrage über die Sat-Search-API auch ein Satellitenbild gefunden hat.
Weiterhin wird der *User-Input* durch *try* und *except* Blöcke zum einen auf Vollständigkeit und zum anderen auf Korrektheit überprüft um eventuelle Eingabefehler abzufangen. Ebenfalls wird für das *Error-Handling* in Randfällen auf einen *try* und *except* Block zurückgegriffen, um zu gewährleisten, dass die Daten auch korrekt gelesen und weitergegeben werden.

6. Optimize Software Only after It Works Correctly

Diese Prämisse wurde während der Entwicklungsphase leider verletzt. Für eine Performancesteigerung wurden sowohl die *custom_tiled_calc*- als auch die *optimal_tiled_calc*-Funktion parallelisiert. Jedoch wurde erst nach der Parallelisierung klar, dass das Skript nicht funktioniert, wenn die beiden Input-Bilder eine unterschiedliche Shape besitzen. Dahingehend musste im Nachhinein ein Zwischenschritt zum *Resamplen* der einzelnen *Tiles* eingebaut werden. Weiterhin wurde festgestellt, dass bei einer unzureichenden Schnittmenge der beiden Satteliten-Bilder aufgrund der Interpolation, die zum *Resamplen* der *Tiles* verwendet wird, ein zu ungenaues Ergebnis entsteht. Deshalb wurde dem *Repository* ein zweites Skript hinzugefügt, welches die NDVI-Differenz näherungsweise nur für die Schnittfläche der beiden Satellitenbilder berechnet. Dieses kann verwendet werden, falls sich die Satellitenbilder nur marginal schneiden.

Nach Wilson et al. (2017) sollte, bevor Code selbst geschrieben wird, zuerst auf bereits existierende Bibliotheken zurückgegriffen werden. Mit *Rio-Mucho* existiert zwar bereits eine Bibliothek für das parallele verarbeiten von *Rasterio-Windows*, jedoch erwies es sich als wesentlich funktionaler eine angepasste Version des *Concurrent Processings* aus der *Rasterio*-Dokumentation zu implementieren.

Weiterhin wurden im Rahmen des Optimierungsprozesses der Datentyp der einzelnen Arrays und der Output-Datei auf Float32 geändert. Dies sollte die Rechengeschwindigkeit weiterhin erhöhen, da nun weniger Bytes an die CPU übergeben werden müssen. Über den Befehl rasterio.dtypes.get_minimum_dtype() wurde festgestellt, dass um die NDVI Ergebnisse noch korrekt darzustellen der minimale Datentyp Float32 beträgt. Zudem emuliert Python alle kleineren Datentypen, wie Float16, weshalb es zusätzlich nicht ratsam ist aus Optimierungsgründen einen Datentyp kleiner als Float32 zu wählen. Ebenfalls wurden für alle Berechnungen *Numpy*-Operatoren bzw. Arrays verwendet, da diese vektorisiert sind und so das Von Neuman Bottle-Neck gemindert werden kann.

Insgesamt wurde durch die Optimierung eine Performance Steigerung von 169 Sekunden (Faktor ca. 2,5) für das *Tiling* mit der internen Blockstruktur und 330 Sekunden (Faktor ca. 1,9) für das Benutzerdefinierte *Tiling* erreicht (Tabelle 1).

Tabelle 1: Laufzeiten des Skriptes mit und ohne Optimierung für die gleichen Datensätze. Die Tile-Size für das Custom-Tiling betrug 100000x100000m. Quelle: Eigene Darstellung.

	Optimal Tiling	Custom Tiling
Normal	279 Sekunden	685 Sekunden
Optimiert	110 Sekunden	355 Sekunden

Es ist anzumerken, dass die Optimierung noch nicht perfekt ist, da gerade für jedes *Window* ein Datenset geöffnet werden muss, wodurch einiges an *Overhead* entsteht.

7. Document Design and Purpose, Not Mechanics

In Kapitel 2 wurde bereits auf die verwendeten *Docstrings* eingegangen. Zusätzlich zu diesen wurden einzelnen Code-Segmenten noch erklärende Kommentare beigefügt. Die Kommentarfunktion wurde ebenfalls genutzt um Code, der nicht selbst geschrieben wurde zu kennzeichnen und die entsprechende Quelle zu vermerken. Darüber hinaus wurde dem GitHub-*Repository* eine *ReadME*-Datei hinzugefügt. In dieser ist die Funktionsweise sowie die Abhängigkeiten des Skriptes erläutert.

Literatur

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., … Wilson, P. (2014). Best Practices for Scientific Computing. PLOS Biology, 12(1).

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. PLOS Computational Biology, 13(6).

Eigenständigkeitserklärung

Hiermit erkläre ich, dass die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt wurde. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Julian Vetter