Heidelberg 6.11.2019

Seminar: Computer Science for geographers 2019 Ruprecht-Karls Universität Heidelberg

Dozent: Christina Ludwig Autor: Clemens Langer(6tes Fachsemester BA)

Eintleitung und Aufbau

Die Grundidee

Aufnahmen der Erdoberfläche mittels der Landsat 8 oder Sentinel 2 Satellitensystemen, erfolgen in regelmäßigem Abstand, was ermöglicht räumlich-zeitliche Phänomene zu betrachten. In diesem Programm wird exklusiv der NDVI betrachtet, welcher mittels einer Normalisierung der Differenz zwischen dem Nah infrarot Bereiches und dem Rot Bereiches einer Aufnahme, Informationen über die vorhandene Vegetation liefert. Analysiert werden Geometrien auf ihren durchschnittlichen NDVI Wert über einen gewissen Zeitraum. Die Geometrien können entweder direkt mittels des GeoJSON Formates gegeben werden oder können für bestimmte Bereiche (ebenfalls mittels GeoJSON) und Filtereinstellungen via OHSOME aus den verfügbaren Open Street Map daten heruntergeladen werden. Der Output lässt sich je nach Wunsch mittels Graphen darstellen und/oder zur Weiterverarbeitung in ein GeoJSON ausgeben.

Der Grundliegende Aufbau

ndvi_main.py - Der Ablaufplan

Das ndvi.main ist für den eigentlichen Ablauf der Analyse verantwortlich, es managet neben der Abfolge der einzelnen Funktionen auch die Input Parameter und die generierung der gewünschten Outputs in Form von Plots und Statistiken. Im ersten Abschnitt werden die aus dem Config File entnommen Input Parameter ausgelesen, bereits gegebene Dateien eingelesen , optional gewünschte OSM Daten per OHSOME API heruntergeladen und für die spezifizierte Boundingbox und Zeitraum die Informationen für die der Analyse zugrunde liegenden Raster Datensätze mittels des externen Moduls Satsearch angefordert.

Der zweite teil der ndvi_main.py führt die eigentliche Analyse aus, zu beginn werden aus den Features welche noch als Dictionary vorliegen mittels Shapely Geometrien erstellt. Anschließend wird über die einzelnen Rasterdatensätze iteriert. Anschließend werden die benötigten Bändern für die Berechnung des NDVI's heruntergeladen, der NDVI berechnet, eine Liste für jede Geometrie erstellt, welche die Indizes der Raster Zellen welche sich mit der Geometrie schneiden beinhaltet und abschließend wird der Durchschnitt der Werte der Indizes für jedes Feature berechnet.

Der Abschließende Teil beinhaltet die beiden möglichen Outputs. Zum einen können alle gesammelten NDVI Werte für jedes Feature gegen den jeweiligen Timestamp mittels matplotlib plottetet werden. Alternativ/Zusätzlich kann ein GeoJSON erstellt werden, mit dem NDVI werten und ihren zugehörigen Timestamp.

Heidelberg 6.11.2019

Seminar: Computer Science for geographers 2019 Ruprecht-Karls Universität Heidelberg

Dozent: Christina Ludwig Autor: Clemens Langer(6tes Fachsemester BA)

utils_geojson.py – Alles rund um GeoJSON

Die Funktionen der utils_geojson.py waren bereits im Vorhinein vorhanden und wurden für dieses Projekt wiederverwendet. Einerseits können mittels eines Dateipfades die Features eines GeoJSONs ausgelesen werden, sowie eine Boundingbox um eine Gruppe von GeoJSONS gelegt werden. Dieser Teil beinhaltet eine Auswahl von Funktionen die bei der Arbeit mit GeoJSONs öfters gebraucht werden können.

Utils.py – Ausführung der eigentlichen Analyse Schritte

Die utils.py beinhaltet die wesentlichen Funktionen des Programms und alle Funktionen wurden erst für dieses Programm gebaut. Ein Block an Funktionen beinhaltet das Reprojizieren von Punkten, Boundingboxen und Geometrien, diese werden teilweise zur Ausführung der Restlichen Funktionen benötigt. Außerdem sind zwei Funktionen zum herunterladen der benötigten Raster-Daten enthalten, eine ließt die benötigten URLs aus und die andere lädt die Daten für den Bereich der Boundingbox in ein Numpy Array und liefert zusätzliche Informationen, wie das verwendete Koordinaten Bezugs System oder eine Matrix zum transformieren von Index zu Koordinaten.

Zusätzlich dient eine Funktion dazu den NDVI für die herunter geladenen Numpy Arrays zu berechnen und 2 Funktionen um die Geometrien mit dem NDVI Array zuschneiden und die Werte herauszulesen. Die Meisten Funktionen bauen also direkt aufeinander auf und bedingen sich teilweise, weshalb ein zusammenfassen innerhalb eines Moduls als sinnvoll erachtet werden kann.

Vorgehen und bekannte Probleme

Verwendete best und good enough practices

Während des Bearbeitens der Aufgaben wurden sowohl best (Wilson et al., 2014), als good enough practices (Wilson et al., 2017) genutzt. Im Folgenden liste ich verwendete Practices auf.

Die Verwendeten best Practices (die *Kursiv* geschrieben wurden unter angaben von gründen nicht beachtet)

1. **Write programs for people, not computers.**
   a. **A program should not require its readers to hold more than a handful of facts in memory at once.**

      Durch die Struktur des Programms und einer intuitiven Benennung sollte eine Klar nachvollziehbarer Ablaufplan des Scripts gegeben sein

   b. **Make names consistent, distinctive, and meaningful.**

      Die einzelnen Namen wurden so gewählt, dass sich ihre Funktion direkt erschließt

c. **Make code style and formatting consistent.**

der Pep8 Standard wurde weitestgehend eingehalten, unter Verwendung von Pylint

2. **Let the computer do the work.**

a. **Make the computer repeat tasks.**

Gegeben durch die Einteilung in Funktionen

b. **Save recent commands in a file for re-use.**

Durch das benutzen eines Config Files und abspeichern jener, kann das programm immer wieder unter den gleichen inputs gestartet werden.

c. **Use a build tool to automate workflows.**

Eine Automation liegt durch den Code und seine Funktionen vor

3. **Make incremental changes.**

a. **Work in small steps with frequent feedback and course correction.**

b. Use a version control system.

Die Nutzung von Git war meiner Ansicht nach nicht von Nöten, da das Script innerhalb einer recht kurzen Zeitspanne entstand

4. **Don't repeat yourself (or others).**

a. **Every piece of data must have a single authoritative representation in the system.**

Durch abkapseln der einzelnen Funktionen ist dieser Teil erfüllt

b. **Modularize code rather than copying and pasting.**

Gegeben durch die Struktur des Programmes

c. **Re-use code instead of rewriting it.**

Funktionen und Teilweise deren aufbau, sind aus vorherigen Projekten übernommen, die Funktionen des utils_geojson waren beispielsweise bereits vorhanden.

5. **Plan for mistakes.**

a. **Add assertions to programs to check their operation.**

Verschiedene Asserts wurden, unter anderem zum überprüfen der Korrektheit der Input Parameter verwendet

b. **Use an off-the-shelf unit testing library.**

Es wurde Unit Test verwendet

   c. **Use a symbolic debugger.**

      Pycharm beinhaltet einen symbolic debugger

6. **Optimize software only after it works correctly.**
   a. **Write code in the highest-level language possible.**

      Das Projekt wurde mittels Python gecoded

7. **Document design and purpose, not mechanics.**
   a. **Document interfaces and reasons, not implementations.**

      Diese Ausführung und Informationen sollten einen Überblick über Funktion und Gebrauch des Programmes liefern

   b. **Refactor code in preference to explaining how it works.**

      Die genaue Funktion des Codes ist durch die Benennung und Struktur erschließbar. Es wurde versucht komplexere schritte in kleiner zu unterteilen, damit sollte der Code auch ohne große Erklärung verständlich sein

   c. **Embed the documentation for a piece of software in that software.**

      Docstrings wurden für die einzelnen Funktionen erstellt

Verwendete Good enough practices (ohne wiederholung aus den best practices)

1. *Datamanagement*
2. **Software**
   a. **Always search for well-maintained software libraries that do what you need.**

      Einige der Funktionen, wie das finden und herunterladen der Daten, die Arbeit mit den Geometrien(Shapely) oder Raster-Daten(Rasterio) übernommen

   b. **Make dependencies and requirements explicit.**

      Import Statements im Code, sowie requirements.txt für die im Projekt genutzten Module.

   c. **Do not comment and uncomment sections of code to control a program's behavior.**

   d. **Provide a simple example or test data set**

Ein Beispiel mit heidelberg.json und heidelberg.geojson wurde bereitgestellt

3. *Collaboration*.
4. **Project organization**
   a. **Put each project in its own directory, which is named after the project.**
   b. **Put text documents associated with the project in the doc directory.**

      README.md
   c. **Put raw data and metadata in a data directory and files generated during cleanup and analysis in a results directory.**

      Siehe Output(standard) ordner
   d. **Put project source code in the src directory.**

      Src Ordner wurde angelegt
   e. **Name all files to reflect their content or function.**
5. **Keeping track of changes**
   a. **Back up (almost) everything created by a human being as soon as it is created.**

      Pycharm speichert die letzen schritte automatisch.
6. **Manuscripts**
   a. **Write manuscripts using online tools with rich formatting, change tracking, and reference management.**

      Readme.md (markdown) und die Ausarbeitung
   b. **Write the manuscript in a plain text format that permits version control.**

      Readme.md (markdown) und die Ausarbeitung

Bekannte Probleme:

Sentinel 2 Daten und Satsearch:

Satsearch liefert zum download der benötigten Sentinel 2 Daten Urls. Diese Funktionieren Momentan jedoch nicht, da sich die benötigten Daten auf Amazon AWS hinter einer Paywall befinden. Eine Lösung könnte eine andere Bezugsquelle, wie sentinel hub liefern. Momentan können zur Analyse nur Landsat 8 Daten benutzt werden.

Boundingboxen um Polygone:

Wenn zwei weit entfernte Polygone in das Programm eingegeben werden kann es zu Problemen und unnötigem Rechenaufwand kommen. Stand jetzt eignet sich das Programm nur für

zusammenhängende Input Features, Bsp Stadtteilgrenzen. Hier für bieten sich gleich zwei Lösungen an:

1. Überprüfen ob ein Mittels satsearch gefundenes Tile sich überhaupt mit den Features schneidet und selektieren derjenigen Features die sich innerhalb Tiles, bevor die Maske erstellt wird
2. Aufteillen von nicht räumlich zusammenhängenden Polygonen vor der Prozessierung

Diese beiden Lösungen könnten unnötige Rechenarbeit ersparen.

Literatur

WILSON, G. ET AL. (2014): Best Practices for Scientific Computing. PLOS Biology, 12(1)

WILSON, G. ET AL. (2017): Good enough practices in scientific computing. PLOS Computational Biology, 13(6)