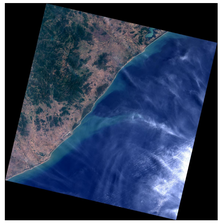
FMask for Python

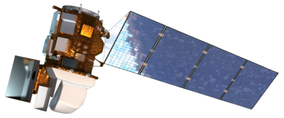
..with an eye towards machine learning

NASA Earth Exchange

CENTER FOR APPLIED ATMOSPHERIC RESEARCH AND EDUCATION

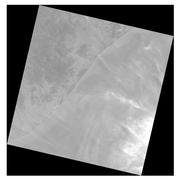
Motivation



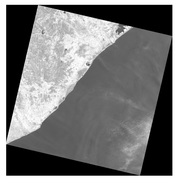


Landsat8 orbits the planet, scanning images of its surface. It scans in 12 spectral bands, ranging from visible to infrared to thermal light. (Left: a Landsat8 scene featuring part of India's eastern seaboard)

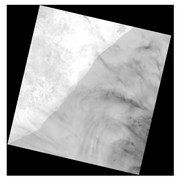
Pictured below are three such spectral bands in the scene:



Blue
(0.452 - 0.512 µm)



Shortwave infrared
(1.566 - 1.651 µm)



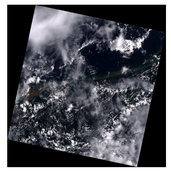
Thermal infrared
(10.60 - 11.19 µm)

We can use these different bands to make suppositions about the surface. For example, we know that clouds are colored white, but so are other surfaces (snow, ice, buildings, rocks, etc). E.g. we use more than the visible spectrum. For example, clouds have low response in the

Landsat8 scenes are useful in applications like Google Earth, deep learning, climate prediction, etc. But for patterns like land development, deforestation, etc. we have a problem....

Clouds get in the way!

FMask aims to mitigate that problem by answering the following question:



How much are clouds influencing sensor readings at any given point?

Andrew Kalenda
Author

Dr. Sherry Palacios
Coordinator

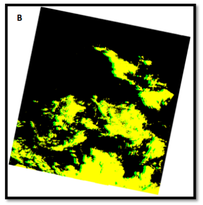
Dr. Weile Wang
Mentor, sponsor

Dr. Sen Chiao
CAARE Director

Considerations

Fmask is an algorithm designed for Landsat.
It combines sensor data from all of the bands to determine:

- Which pixels are clear?
- Which pixels are clouded?
- Which pixels are shadowed by clouds?
- Which pixels are snow?



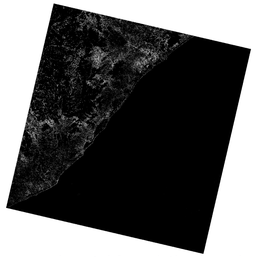
Its creation is described in:

Zhu, Zhe, Shixiong Wang, and Curtis E. Woodcock. "Improvement and expansion of the Fmask algorithm: cloud, cloud shadow, and snow detection for Landsats 4–7, 8, and Sentinel 2 images." Remote Sensing of Environment 159 (2015): 269-277.

Zhu, Zhe, and Curtis E. Woodcock. "Object-based cloud and cloud shadow detection in Landsat imagery." Remote Sensing of Environment 118 (2012): 83-94.

But its current design does not fill all of our needs. We need an implementation that fits in our Python pipeline.

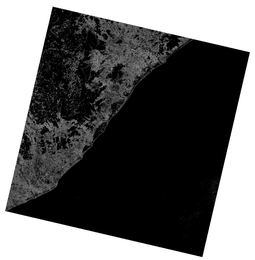
We need more flexibility than the original FMask algorithm provides. We need to be able to experiment with intermediary and ancillary calculations, and get the results.



Experimental formulation based on human perception of brightness

For example, one of the steps FMask takes is performing a "whiteness test." It's far from the final product, but we may want to experiment with it. Our implementation should provide the means to do so easily.

Perhaps most importantly, our implementation is not just the algorithm as described in the paper. It is a library - a suite of tools - that can be used programmatically by future machine learning algorithms.



FMask's "whiteness test" original formulation

There are many details in the implementation that come from this. Notably:

- Magic numbers as "hyperparameters":
- FMask function implemented in a "functional" programming style:
- Vector-based calculations:

Design Decisions

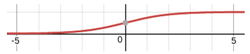
The code base is divided into two parts:

- FMask.py, which is a library providing a suite of data-transformation functions which, in sequence, comprise the FMask algorithm.
- Landsat8Scene.py, which handles reading from and writing to GeoTIFF data files, as well as various convenience methods for downloading a scene from Amazon Web Services and for using FMask.py in its typical fashion.

This gives room to grow in the future, as modules like Landsat8Scene.py can be created in the future to handle other specific data sources. For example, the Harmonized Landsat Sentinel-2 dataset that, as of this poster's making, is soon to come.

Another significant design decision is dropping the algorithm's boolean masks in favor of "confidence" layers. A confidence value is a number ranging from 0 to 1 that rates the strength of an assertion. This sounds generic because it is. We are interested in it because it unifies probabilities with percentages into a single data type.

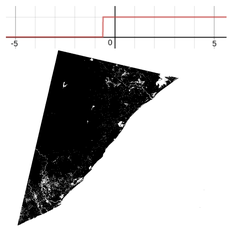
Where before we may make a comparison resulting in a boolean value, we now use an activation function resulting in a confidence value. A near-zero value is analogous to false, and a near-one value is analogous to true.



The most common activation function: the logistic function

$$f(x) = \frac{1}{1+e^{w(l-g)}}$$

with w=1, l=0, and g as the independent variable



A scene valuing confidence in there being water, with w=0, l=0, and g as the NDVI (normalized difference vegetation index). The closer a pixel gets to 1, shown as white, the stronger the assertion that it is water and not land.

These can still be used in a familiar, boolean-like way simply by setting a very steep curve (as demonstrated at left).

Nonetheless, the data remains numeric and without need of type-casting.

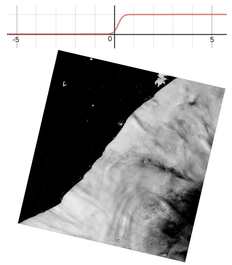
Vector lengths have no need to expand or contract, as they are not being masked out -- only assigned relative strengths to be factored into calculations.

We can frame assertions as confidence values by carefully selecting what values to use in the activation function.

In the above example, there are many false positives shown as white speckles. A more shallow curve will reflect ambiguity near the threshold value (l).

Combining a scaling factor (w) with the threshold value can neatly frame the curve to capture a desired view. In the example at right, the curve has been set such that partial cloud coverage is reflected in shades of grey.

These confidence values can then be used in other formulae to give relative weights to individual pixels.



The same scene valuing confidence in there being water unobstructed by clouds, with w=9, l=0.2, and g as NDVI. The gradations from black to white capture both the extent to which clouds of varying thickness cover the water, and uncertainties in our measurements and calculations.

Getting Started

This implementation of FMask is currently hosted in a public Github repository:

http://www.github.com/akalenda/PyFMask

The repository contains instructions on installation and basic usage, both of which are intended to be easy for anyone with basic programming proficiency. Python3 is required.

Our library makes use of some 3rd-party libraries, which must be installed according to their own user manuals: Theano, Rasterio, and Pandas. These may change in future versions of FMask for Python.

Future Work

From code optimizations, to research on functionality, to followup machine learning applications, there is much to do! Visit the Github Issues page to learn more.