Table of Contents

Software Requirements

- TDM-CC
- Win32 OpenSSL
- ArcGIS Desktop
- R
- Python 2.7 **32 bits**
- Python packages:
  - setuptools
  - virtualenv
  - google-api-python-client
  - pyopenssl
  - dbfpy
  - numpy (comes with ArcGIS)
  - arcpy (comes with ArcGIS)

Setup

Note: commands starting with > are run in the command prompt, commands starting with >>> are run in python.

Installation

1. Install TDM-CC, OpenSSL, ArcGIS Desktop and R. Instructions for TDM-CC:

http://www.megustaulises.com/2013/07/unable-to-find-vcvarsallbat_9.html

OpenSSL binaries, install the 32 bits full (not light) version:

http://slproweb.com/products/Win32OpenSSL.html

2. Add the following to the PATH environment variable (change version numbers as needed):

C:\Python27\ArcGIS10.1\;C:\Python27\ArcGIS10.1\Scripts;C:\OpenSSL-Win32\bin;C:\MinGW32\bin;C:\Program Files\R\R-3.0.1\bin

3. Install setuptools. If there is more than one version of python installed make sure to use the python.exe that comes with ArcGIS.

The command:

```
> for %i in (cmd.exe) do @echo.   %~$PATH:i
```

should return ‘C:\Python27\ArcGIS10.1\python.exe’, if not the PATH environment variable need to be changed so that only the ArcGIS version of Python is in PATH, or in all commands in the rest of this document ‘python’ should be replaced by the full python path, e.g.:

```
> C:\Python27\ArcGIS10.1\python.exe ez_setup.py
```

instead of

```
> python ez_setup.py
```

Download and extract setuptools, go to the folder where setuptools was extracted and run:

```
> python ez_setup.py
```

4. Install virtualenv. Download and extract virtualenv, and in the folder where virtualenv was extracted, run:

```
> python setup.py install
```

5. Create python virtual environment. The virtual environment allows installing packages in an isolated environment to prevent messing up the ArcGIS python install or messing up ALEM when ArcGIS's python is updated.

Go to the location where you want the virtual environment folder to be created and run:

```
> cd C:\path\to\where\we\want\to\install\virtualenv
> virtualenv --system-site-packages alem-virtualenv
```

where the second argument, 'alem-virtualenv', is the name of the virtual environment folder (could use a different folder name but make sure to use the right folder in other commands).

The --system-site-packages switch makes system packages available in the virtual environment, we need this since we use arcpy and numpy, which are installed with ArcGIS.

6. Activate virtual environment.

In the same location as step 5:

```
> .\alem-virtualenv\Scripts\activate.bat
```

7. Install packages. google-api-python-client and pyopenssl are found in the PyPI repository and can be installed as follows:

```
> pip install google-api-python-client
> pip install pyopenssl
```

dbfpy is not in PyPI and must be installed manually. Download and extract, in the folder where dbfpy was extracted run:

```
> python setup.py install
```

8. Sanity check. Should be able to import these modules successfully:

```
> python
>>> import google-api-client
>>> import dbfpy
>>> import arcpy
```

Settings

Most of the settings for ALEM can be found in settings.py and are self-explanatory. Settings include the e-mail address of the service account, the filenames and folder structure of input and output files, table IDs for fusion tables, default analysis settings,location of scripts, and so on.

Fusion Tables API

The application is authorized to use the Google APIs through a service account. The service account is managed through the Google API Console: https://code.google.com/apis/console

Fusion tables that are modified by the application need to be editable by the service account: 269347469410-gn6k5pjdekk1t5h8pj9hrmt8agsbkbf5@developer.gserviceaccount.com.

However ownership should not be given to the service account since we cannot login with it and it will become impossible to change the owner back to a user account, and any functionality that is accessible through the web interface but not the API will be inaccessible.

Documentation for Fusion Tables API Python Library: https://google-api-client-libraries.appspot.com/documentation/fusiontables/v1/python/latest/

Documentation for Fusion Tables API: https://developers.google.com/fusiontables/

Notes on Scripts

Python Scripts

alem.py

Used to process images automagically or manually from the command line - in development/not functional as of 7/22/2013. The goal is for alem.py to automatically download new images and process them, upload results, etc. with no user interaction needed.

parallelalem.py
Used to process images in parallel - can be used to run any method from ALEMObject class on multiple images from the command line (see Usage below).

alemobject.py
Defines the ALEMObject class.

Each Landsat image being analysed is associated with an ALEMObject instance. The instance's variables contain the analysis settings (which band ratios, polygon or point zonal statistics, etc.), image metadata, and the analysis state (what is the next step).

The state of the instance is saved on disk with the Pickle module so that ALEM can 'remember' the analysis state and analysis settings even if the program is interrupted.

The ALEMObject class also defines the different methods used to analyse images. The methods execute various tasks on the files of the image associated with the ALEMObject instance - for example, generating band ratios from the band TIF files.

alemutils.py
While alemobject.py defines the functions used for image analysis, alemutils.py defines other useful functions that are more 'external'.

Generally the functions in alemobject.py work on the landsat image files associated with an ALEMObject instance, while the functions do things like creating new ALEMObject instances, downloading new images, running analyses on multiple instances, and so on.

apifunctions.py
This file defines the FusionTable class, which contains useful methods to interact with the Google Fusion Tables API. The methods are called by other ALEM functions and do not need to be called directly.

utilfunctions.py
This file defines some other useful functions called by ALEM and which are also not called directly.

R Scripts

afn.kappa-tau-taup-taue.R

leaps.R

merge_sel_tables.R

merge_tables.R
R script that merges tables containing band ratios and CDOM values; the merged tables are used as input for regression.R.

merge_tables_all_bands.R

merge_tables_raw.R

regression.R
R script that computes the linear regression model between bands and CDOM/DOC. Produces predictions for all lakes in the image as well as a pdf of the regression curve.

regression_no_pred.R

select_training_set.R

switcheroo.R

tree_prediction.R

tree_prep.R

tree_testing.R

Usage
First, activate the python virtual environment:

```
> path\to\virtualenv\alem-virtualenv\Scripts\activate.bat
```

Using the Command Prompt

Multiple Images in Parallel

Using Python

Python Functions

alemobject methods

configure_analysis(kwargs)**	Configure analysis parameters. Parameters can be set by optional keyword arguments (bqa_or_rad, zstat_mode, var, b, bdivb, bxb), else the defaults values from settings.py are used.
create_folders()	Create folder hierarchy (only creates folders that are missing, does not overwrite existing files or folders).
parse_metadata()	Parse metadata file from Landsat Level 1 Product.

upload_metadata()	Upload selected metadata to fusion table (list of processed images).
set_up_default()	Configure analysis with default parameters, create folders, parse and upload metadata.
arcgis_prepare_toa()	Prepare top-of-atmosphere (TOA) corrected rasters with ArcGIS for bands specified in the analysis parameters.
arcgis_prepare_bqa()	Prepare BQA corrected rasters with ArcGIS (cloudy pixels are removed).
arcgis_prepare_band_ratios()	Prepare band ratio rasters as specified in analysis parameters.
arcgis_prepare_band_products()	Prepare band product rasters as specified in analysis parameters.
arcgis_zstat_poly_analysis()	Compute zonal statistics of lake polygons for each band, band ratio and band product raster as specified in analysis parameters.
arcgis_zstat_points_analysis()	Compute zonal statistics of sample polygons (buffered area around each lake sample) for each band, band ratio and band product raster as specified in analysis parameters.
arcgis_analysis()	Automatically prepare all rasters and do zonal statistics based on analysis parameters.
convert_dbfs_to_csvs()	Convert all dbf files output by ArcGIS to csv files.
merge_points_csvs()	Combine individual sample point csvs into one csv containing band values for all sample points in the image.
get_sample_data()	Download sample data (CDOM or DOC) for the image from the lake samples Fusion Table and write it to csvs.
r_merge_poly_csvs()	Merge sample data csvs and bands csvs for lake polygons.
find_band_minimums()	Get the minimum value for each band in the image for Dark Object Substraction (DOS) correction.
r_compute_regression()	Compute linear regression from merged csvs (either polygons or points) containing band values and sample data.
r_leaps_regression()	Compute regressions for multiple subsets of bands/band ratios/band products to find the combination yielding the best model.

r_analysis()	Convert dbfs, get sample data, merge csvs and compute regression.
update_running_estimates()	Upload DOC/CDOM estimates of lakes (by NID) obtained from the regression to fusion table, and compute a running estimate for each lake based on new and past predictions.
clean_up()	Deletes dbf files and rasters associated with the image.
run_next()	Runs the next analysis step for the image based on the last recorded step (for example, would run upload_metadata() if the last method ran was parse_metadata()).

alemutils functions

find_new_images_in_metadata()	Find new images in Landsat bulk metadata file (http://landsat.usgs.gov/consumer.php).
init_image(sceneId)	Create AlemObject instance, pickle it and create folder structure for new image.
recreate_object(sceneId)	Recreate AlemObject associated with scene id and writes it to a pickle object (overwrites existing file).
update_instance(scene)	Returns updated AlemObject. Parameter 'scene' can be a scene id string or an alemobject instance. Must be run to use updated code with previously existing alemobjects.
list_scenes()	Returns a list of all scenes that are 'in process' (for which a folder exists in the image analyses folder).
load_scene(sceneId)	Loads the pickle file associated with a scene ID and returns its alemobject.
pickle_object(alemObject)	Save an alemobject as a pickle file.
combined_regression(imagesList)	Concatenate csv files and do a combined regression for all images in the list. The parameter imagesList is the filename of the txt file with the list of scene ids to include (located in the Combined_Images folder).
r_leaps_combined()	Do a LEAPS regression on multiple images (see above).
convert_dbfs_to_csvs(folder)	Convert all dbf files in a given folder to csv format.
check_state(imagesList)	Produces a csv file detailing the state of analysis of all images listed in imagesList (txt file with list of scene IDs).

Appendix

Links to Fusion Tables

List of processed images, with KML geometry (updated by ALEM)

All sample data

SuperDOC only sample data

Running CDOM estimates by NID

Number of lakes and samples by path/row/scene (generated by ALEM)

Links to Google Spreadsheets

List of processed images, with analysis details (updated manually)

Number of samples in each tile by path/row

Number of lakes in each tile by path/row

Summary of number of lakes and samples by path/row