YATSM

Release 0.7.0

Sep 28, 2017

Contents

CHAPTER 1

About

Yet Another Timeseries Model (YATSM) is a Python package for utilizing a collection of timeseries algorithms and methods designed to monitor the land surface using remotely sensed imagery.

YATSM can be thought of as three components:

1. A library of functions and other objects useful for performing time series analysis, including various statistics and input/output helpers.
2. Time series analysis algorithm implementations
3. The command line interface (CLI) provides a set of programs that facilitate the running of time series algorithms on "time series stacks" and the analysis of the results.

CHAPTER 2

User Guide

To get started with YATSM, please follow this user guide:

Installation

Dependencies

Yet Another TimeSeries Model (YATSM) is written in Python (version 2.7, but 3.5 is soon to be a priority) and utilizes several C libraries, including the Geographic Data Abstraction Library (GDAL). Package dependencies are most easily installed using Conda, but may also be installed using your package manager on Linux (see *Ubuntu 14.04* example), Homebrew on Mac OS, or through OSGEO4W on Windows.

However, it is strongly encouraged that you install YATSM into an isolated environment, either using virtualenv for `pip` installs or a separate environment using conda, to avoid dependency conflicts with other software.

Conda

Requirements for YATSM may also be installed using conda, Python's cross-platform and platform agnostic binary package manager from ContinuumIO. conda makes installation of Python packages, especially scientific packages, a breeze because it includes compiled library dependencies that remove the need for a compiler or pre-installed libraries.

Installation instructions for `conda` are available on their docs site conda.pydata.org

Since conda makes installation so easy, installation through conda will install all non-developer dependencies. Install YATSM using conda into an isolated environment by using the `environment.yaml` file as follows:

```
# Install
conda env create -n yatsm -f environment.yaml
# Activate
source activate yatsm
```

And as with `pip`, you need to install `YATSM`:

```
# Install YATSM
pip install .
```

virtualenv and pip

virtualenv

Python and GDAL are usually installed on most computing environments that handle geographic data (if not, see an example of installing these dependencies on *Ubuntu 14.04*). If you wish to install YATSM on a system with these two dependencies but you don't have installation privileges for the Python dependencies, you could install YATSM into a virtualenv.

virtualenv creates isolated Python environments, thus enabling a user without root privileges to install modules. *virtualenv* has the added benefit of enabling the installation of specific module versions for single piece of software.

To set up a *virtualenv*, it must first be available. With root access,

```
sudo pip install virtualenv
```

Once *virtualenv* is installed, users may create a *virtualenv* for YATSM:

```
virtualenv yatsm_venv
```

To activate this isolated Python environment from Bash:

```
source yatsm_venv/bin/activate
```

Your terminal prompt will change, denoting the switch to this newly created isolated Python environment.

pip

The basic dependencies for YATSM are included in the `requirements.txt` file which is by `pip` as follows:

```
pip install -r requirements.txt
```

Additional dependencies are required for some timeseries analysis algorithms or for accelerating the computation in YATSM. These requirements are separate from the common base installation requirements so that YATSM may be more modular:

- Long term mean phenological calculations from Melaas *et al.*, 2013
  - Requires the R statistical software environment and the `rpy2` Python to R interface
  - `pip install -r requirements/pheno.txt`
- Computation acceleration
  - GLMNET Fortran wrapper for accelerating Elastic Net or Lasso regularized regression
  - Numba for speeding up computation through just in time compilation (JIT)
  - `pip install -r requirements/accel.txt`

A complete installation of YATSM, including acceleration dependencies and additional timeseries analysis dependencies, may be installed using the `requirements/all.txt` file:

```
pip install -r requirements/all.txt
```

Finally, install YATSM:

```
# Install YATSM
pip install .
```

Developer Installation

If you're interested in helping develop YATSM, you can download the repository using Git and install it in an editable installation:

```
git clone https://github.com/ceholden/yatsm.git
cd yatsm/
pip install -e .
```

Documentation may be built using Sphinx from the *docs* directory:

```
cd docs/
make html
```

Platform Support

YATSM is developed on Linux (CentOS 6, Arch, and Ubuntu 14.04) and has not been tested on any other platforms, though I have seen it working on Mac OS. I am welcome to any help fixing bugs or better supporting Windows, but I will not try to support Windows myself.

Ubuntu 14.04

On Ubuntu 14.04, for example, the GDAL build dependencies may be satisfied by installing the following:

```
sudo apt-get install python2.7-dev
sudo apt-get install gdal-bin libgdal-dev
sudo apt-get install python-gdal
```

This installation guide will also utilize the pip utility for installing Python modules. *pip* may be installed or upgraded following the instructions here, but it is usual preferable to use your package manager:

```
$ sudo apt-get install python-pip
```

You will also need two font packages for *matplotlib* that are not installed with Ubuntu by default:

```
$ sudo apt-get install libfreetype6-dev libxft-dev
```

With the GDAL library and *pip* installed, follow the guide for how to install YATSM below using *virtualenv and pip*.

Dataset Preparation

This part of the guide is a work in progress. For now you can get an idea of how the datasets should be organized and prepared by viewing two example datasets located here.

The issue tracker on Github is being used to track additions to this documentation section. Please see issue 34.

Model Parameter Exploration

The issue tracker on Github is being used to track additions to this documentation section. Please see issue 35.

Timeseries Model Specification

This section will describe the timeseries models used in YATSM and how to configure them *within the configuration file*.

The issue tracker on Github is being used to track additions to this documentation section. Please see issue 36.

Science Models

The time series model run using the YATSM package command line interface (i.e., *yatsm line* and *yatsm pixel*) is specified in the configuration file using the `YATSM['algorithm']` key:

```
YATSM:
    algorithm: "AN_ALGORITHM"
    ...

AN_ALGORITHM:
    init: ...
    fit: ...
```

The value specified for the `algorithm` key within the `YATSM` section must be the name of another section of the configuration file that can be used to identify, initialize, and run a time series algorithm included in the `yatsm.algorithms` module.

Models

CCDCesque

The YATSM implementation of Continuous Classification and Change Detection (CCDC) *[1]*

class `yatsm.algorithms.ccdc.`**CCDCesque**(*test_indices=None*, *estimator={'object': <MagicMock name='mock.linear_model.Lasso()' id='139994854590480'>, 'fit': {}}*, *consecutive=5*, *threshold=2.56*, *min_obs=None*, *min_rmse=None*, *retrain_time=365.25*, *screening='RLM'*, *screening_crit=400.0*, *remove_noise=True*, *green_band=1*, *swir1_band=4*, *dynamic_rmse=False*, *slope_test=False*, *idx_slope=1*, ***kwargs*)

Initialize a CCDC-like model for data X (spectra) and Y (dates)

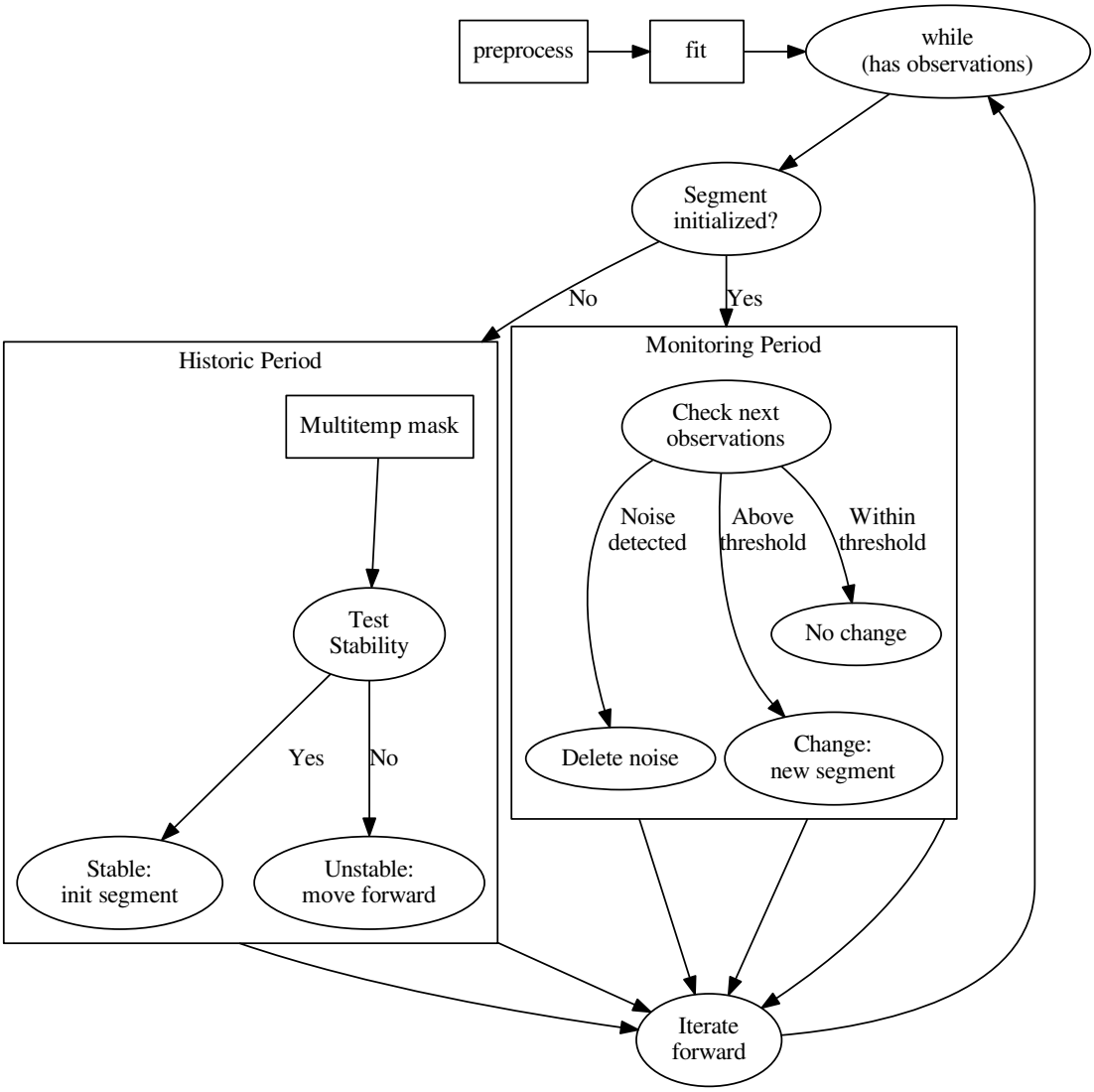
An unofficial and unvalidated port of the Continuous Change Detection and Classification (CCDC) algorithm by Zhu and Woodcock, 2014.

Parameters

- **test_indices** – Test for changes with these indices of `Y`. If not provided, all series in `Y` will be used as test indices

- **estimator** – dictionary containing estimation model from `scikit-learn` used to fit and predict timeseries and, optionally, a dict of options for the estimation model `fit` method (default: `{'object': Lasso(alpha=20), 'fit': {}}`)
- **consecutive** – Consecutive observations to trigger change
- **threshold** – Test statistic threshold for change
- **min_obs** – Minimum observations in model
- **min_rmse** – Minimum RMSE for models during testing
- **retrain_time** – Number of days between model fit updates during monitoring period
- **screening** – Style of prescreening of the timeseries for noise. Options are 'RLM' or 'LOWESS' (default: RLM)
- **screening_crit** – critical value for multitemporal noise screening (default: 400.0)
- **remove_noise** – Remove observation if change is not detected but first observation is above threshold (if it looks like noise) (default: True)
- **green_band** – Index of green band in `Y` for multitemporal masking (default: 1)
- **swir1_band** – Index of first SWIR band in `Y` for multitemporal masking (default: 4)
- **dynamic_rmse** – Vary RMSE as a function of day of year (default: False)
- **slope_test** – Use an additional slope test to assess the suitability of the training period. A value of True enables the test and uses the *threshold* parameter as the test criterion. False turns off the test or a float value enables the test but overrides the test criterion threshold. (default: False)
- **idx_slope** – if `slope_test` is enabled, provide index of `X` containing slope term (default: 1)

Algorithm Logic



Algorithm Flowchart:
CCDCesque

Model Configuration

The issue tracker on Github is being used to track additions to this documentation section. Please see issue 37.

Configuration File

The batch running script uses an YAML file to parameterize the run. The YAML file uses several sections:

1. `dataset`: Describes dataset attributes common to all analysis
2. `YATSM`: Contains model parameters common to all analysis and declares what change detection algorithm should be run. The algorithm specified should be a section within the configuration file.
3. `${ALGORITHM}`: The section referenced as `algorithm` in `YATSM` describing how the time series analysis should be run (e.g., a section `CCDCesque` when `algorithm: "CCDCesque"`)
4. `phenology`: Describes phenology fitting parameters
5. `classification`: Describes classification training data inputs

The following tables describes the meanings of the parameter and values used in the configuration file used in YATSM. Any parameters left blank will be interpreted as `None` (e.g., `cache_line_dir =`).

Dataset Parameters

Note: This section is out of date for *v0.5.0* and requires re-writing

Note: you can use `scripts/gen_date_file.sh` to generate the CSV file for `input_file`.

YATSM Analysis Parameters

Note: This section is out of date for *v0.5.0* and requires re-writing

YATSM Algorithm Parameters

This section will differ depending on what algorithm is used and specified in the `algorithm` key in the `YATSM` section. For more information, visit the *science models guide section* for information the available algorithms.

Phenology

The option for long term mean phenology calculation is an optional addition to YATSM. As such, visit *the phenology guide page* for configuration options.

Classification

The scripts included in YATSM which perform classification utilize a configuration INI file that specify which algorithm will be used and the parameters for said algorithm. The configuration details specified along with the dataset and YATSM algorithm options deal with the training data, not the algorithm details. These training data configuration options include:

Parameter	Data Type	Explanation
`training_data`	`str`	Training data raster image containing labeled pixels
`mask_values`	`list`	Values within the training data image to mask or ignore
`training_start`	`str`	Earliest date that training data are applicable. Training data labels will be paired with models that begin at least before this date
`training_end`	`str`	Latest date that training data are applicable. Training data labels will be paired with models that end at least after this date
`training_date_format`	`str`	Format specification that maps `training_start` and `training_end` to a Python datetime object (e.g., `%Y-%m-%d`)
`cache_xy`	`str`	Filename used for caching paired X features and y training labels

Example

An example template of the parameter file is located within `examples/p013r030/p013r030.yaml`:

```
# Example configuration file for YATSM line runner
#
# This configuration includes details about the dataset and how YATSM should
# run

# Version of config
version: "0.7.0"

dataset:
    # Text file containing dates and images
    input_file: "examples/p013r030/images.csv"
    # Input date format
    date_format: "%Y%j"
    # Output location
    output: "landsat_stack/p013r030/subset/YATSM"
    # Output file prefix (e.g., [prefix]_[line].npz)
    output_prefix: "yatsm_r"
    # Total number of bands
    n_bands: 8
    # Mask band (e.g., Fmask)
    mask_band: 8
    # List of integer values to mask within the mask band
    mask_values: [2, 3, 4, 255]
    # Valid range of band data
    # specify 1 range for all bands, or specify ranges for each band
    min_values: 0
    max_values: 10000
    # Use BIP image reader? If not, use GDAL to read in
    use_bip_reader: False
    # Directory location for caching dataset lines
    cache_line_dir: "landsat_stack/p013r030/subset/cache"

# Parameters common to all timeseries analysis models within YATSM package
YATSM:
    algorithm: "CCDCesque"
    prediction: "sklearn_Lasso20"
    design_matrix: "1 + x + harm(x, 1) + harm(x, 2) + harm(x, 3)"
    reverse: False
    commission_alpha:
    # Re-fit each segment, adding new coefficients & RMSE info to record
```

```
    refit:
        prefix: [rlm_]
        prediction: [rlm_maxiter10]
        stay_regularized: [True]

# Parameters for CCDCesque algorithm -- referenced by "algorithm" key in YATSM
CCDCesque:
    init:  # hyperparameters
        consecutive: 5
        threshold: 3.5
        min_obs: 24
        min_rmse: 150
        test_indices: [2, 3, 4, 5]
        retrain_time: 365.25
        screening: RLM
        screening_crit: 400.0
        slope_test: False
        remove_noise: True
        dynamic_rmse: False
        # Indices for multi-temporal cloud masking (indexed on 1)
        green_band: 2
        swir1_band: 5

# Section for phenology fitting
phenology:
    enable: True
    init:
        # Specification for dataset indices required for EVI based phenology␣
↪monitoring
        red_index: 2
        nir_index: 3
        blue_index: 0
        # Scale factor for reflectance bands
        scale: 0.0001
        # You can also specify index of EVI if contained in dataset to override␣
↪calculation
        evi_index:
        evi_scale:
        # Number of years to group together when normalizing EVI to upper and lower␣
↪percentiles
        year_interval: 3
        # Upper and lower percentiles of EVI used for max/min scaling
        q_min: 10
        q_max: 90

# Section for training and classification
classification:
    # Training data file
    training_image: "training_data.gtif"
    # Training data masked values
    roi_mask_values: [0, 255]
    # Date range
    training_start: "1999-01-01"
    training_end: "2001-01-01"
    training_date_format: "%Y-%m-%d"
    # Cache X feature input and y labels for training data image into file?
    cache_training:
```

Batch Interface

The default method for running image stacks is to run each line, or row, separately from other lines. In a multiprocessing situation, the total number of lines can be broken up among the `n` available CPUs. Before using the batch interface, make sure you already have a parameter file generated as described by *Model Configuration*.

The batch interface which runs each line separately is *yatsm line*. It's usage is:

```
$ yatsm line --help
Usage: yatsm line [OPTIONS] <config> <job_number> <total_jobs>

Options:
  --check_cache  Check that cache file contains matching data
  --resume       Do not overwrite preexisting results
  --do-not-run   Do not run YATSM (useful for just caching data)
  --help         Show this message and exit.
```

Let's say our image stack contains 1,000 rows. If we use 50 total CPUs to process the image stack, then each CPU will be responsible for only 20 lines. To evenly distribute the number of pixels that contain timeseries (e.g., to ignore any NODATA buffers around the images), the lines are divided up in sequence. Thus, job 5 of 50 total jobs would work on the lines:

```
$ job=5
$ n=50
$ seq -s , $job $n 1000
5,55,105,155,205,255,305,355,405,455,505,555,605,655,705,755,805,855,905,955
```

Sun Grid Engine

In the example of the compute cluster at Boston University which utilizes the Sun Grid Engine scheduler, one could run an image stack as follows:

```
$ njob=200
$ for job in $(seq 1 $njob); do
    qsub -j y -V -l h_rt=24:00:00 -N yatsm_$job -b y \
        yatsm -v line --resume config.ini $job $njob
  done
```

By setting the environment variable, `PYTHONUNBUFFERED`, to a nonzero and non-empty value, Python will keep stdout and stderr unbuffered. This is handy to use when trying to diagnose a problem or when trying to guage the progress of YATSM when logging to a file (e.g., in a `qsub` job) because unbuffered output is flushed or written immediately to the log file. Be aware that the constant writing to the log file may incur a penalty cost. You can run YATSM jobs with unbuffered output as follows in the example for `yatsm line`:

```
$ qsub -j y -V -l h_rt=24:00:00 -N yatsm -b y \
    PYTHONUNBUFFERED=1 yatsm -v line --resume config.ini 1 1
```

One useful tip is to optimize the use of the CPU nodes by first transforming the dataset from an image based format to a timeseries format by saving all observations for each row in a separate file. The transformed dataset will be much easier to read as a timeseries because each processor only needs to read in one file instead of finding, opening, seeking through, and reading from many individual image files.

To accomplish this approach, the `--do-not-run` flag can be combined with a specific request for computer nodes with fast ethernet speeds, `-l eth_speed 10`:

```
$ njob=16
$ for job in $(seq 1 $njob); do
    qsub -j y -V -l h_rt=24:00:00 -l eth_speed=10 -N yatsm_$job -b y \
        yatsm -v line --resume --do-not-run config.ini $job $njob
  done
```

Mapping Derived Information

Maps can be created based on the attributes of a timeseries segment for any date desired. These attributes include the statistical model coefficients for each band, the Root Mean Squared Error (RMSE) of the model fits for each band, and the classification value predicted. The predicted reflectance for any band may also be generated using the model coefficients. These maps may be created using the *yatsm map* script.

Missing Values

By default, *yatsm map* will only look for models which intersect the date given by the user. A model intersects a date if the model's starting date is equal to or less than the date and if the model's ending date is equal to or greater than the date. For example, a model beginning on 2000-01-01 and ending on 2002-01-01 intersects 2001-01-01, but this model does not intersect 2002-02-02.

Sometimes it is desirable to produce a classification value or reflectance value even if no models intersect the date provided. For example, we wanted to produce a classification value for 2000-01-01 but the model ended in 1999 and was not re-initialized until 2000-06-01, we might be happy to know what the classification value of the next segment is (i.e., what the pixel turns into).

In other circumstances, we might be okay producing a map using the segment immediately prior to the date of the map provided that there was not a change detected. This is a common case when generating classifications, coefficient maps, or reflectance predictions toward the end of the timeseries.

For these two circumstances, the *yatsm map* script provides the `--before` and `--after` flags. These flags are not mutually exclusive and will not override the default behavior of providing the timeseries segment information which intersects the date provided. Instead, these three choices operate in order of desirability. From most to least desirable:

1. Segment which intersects the date
2. Segment immediately after the date, if `--after` is specified
3. Segment before the date, if the segment does not contain a break and `--before` is specified

The mapped output will always contain the information from the most desirable segment. Behind the scenes, it accomplishes this by providing mapped values for the least desirable option first and then overwriting with more desirable options.

Examples

For example, consider these mapping situations:

1. Create a map showing the image predicted for January 1st, 2000 for all bands

```
$ yatsm map predict 2000-01-01 predict_2000-01-01.gtif
```

2. Create a map showing the image predicted for January 1st, 2000 for all bands, filling in missing values with the information from the timeseries segment immediately after January 1st, 2000, if possible

```
$ yatsm map --after predict 2000-01-01 predict_2000-01-01.gtif
```

3. Create a map showing all the coefficients for the red band (band 3) for January 1st, 2000

```
$ yatsm map --band 3 coef 2000-01-01 coef_red_2000-01-01.gtif
```

4. Create a map of only the time trend, or slope, coefficients for all bands for January 1st, 2000

```
$ yatsm map --coef slope coef 2000-01-01 coef_slope_2000-01-01.gtif
```

5. Create a map of the current land cover, or the land cover that a pixel will turn into, for January 1st, 2000

```
$ yatsm map --after class 2000-01-01 classmap_2000-01-01.gtif
```

Docs TODO

- Example maps
- Images helping explain `--after` and `--before`
- More information on CLI flags / switches
- Explanation of `--root`, `--result`, and `--image` parameters

Mapping Changes

To visualize change information it is often useful to create maps of the date when a pixel changes or to map the total number of changes detected within a desired range of dates. These maps can be created using the *yatsm changemap*.

Examples

For example, consider these mapping situations:

1. For each year from 2000 to 2010, create a map of the first change detected within the year:

```
for y1 in $(seq 2000 2010); do
    y2=$(expr $y1 + 1)
    yatsm changemap first $y1-01-01 $y2-01-01 change_$y1-$y2.gtif
done
```

2. Create a map of the total number of changes, per pixel, for the 2000 to 2010 decade:

```
yatsm changemap num 2000-01-01 2010-01-01 changenumber_2000-2010.gtif
```

Docs TODO

- Example images
- Explanation of `--root`, `--result`, and `--image` parameters

Classification

The issue tracker on Github is being used to track additions to this documentation section. Please see issue 38.

Long Term Mean Phenology

This part of the guide is a work in progress and more information will be added.

The purpose of this module within YATSM is to provide long term mean phenological characteristics of landscapes using an algorithm developed by Melaas *et al* (2013) for each stable land cover segment identified by the CCDC portion of YATSM.

See:

> Melaas, EK, MA Friedl, and Z Zhu. 2013. Detecting interannual variation in deciduous broadleaf forest phenology using Landsat TM/ETM+ data. Remote Sensing of Environment 132: 176-185. http://dx.doi.org/10.1016/j.rse.2013.01.011

Configuration

Example:

```
phenology:
    enable: False
    init:
        # Specification for dataset indices required for EVI based phenology␣
↪monitoring
        red_index: 2
        nir_index: 3
        blue_index: 0
        # Scale factor for reflectance bands
        scale: 0.0001
        # You can also specify index of EVI if contained in dataset to override␣
↪calculation
        evi_index:
        evi_scale:
        # Number of years to group together when normalizing EVI to upper and lower␣
↪percentiles
        year_interval: 3
        # Upper and lower percentiles of EVI used for max/min scaling
        q_min: 10
        q_max: 90
```

Integration of user-provided time series algorithms

The YATSM package provides some time series algorithms as part of the project, but developers can also build and integrate their own algorithms into the suite of time series algorithms that can be utilized by programs like *yatsm line* or *yatsm pixel*.

YATSM uses the setuptools module concept of "entry points" to enumerate time series algorithms. YATSM algorithms are registered into a group based on the type of algorithm – there are entry point groups for algorithms that find change as separate from algorithsm which postprocess existing change results. Both categories link an entry point name to a class or function usable within the YATSM package. For example, from the `setup.py` installation setup script:

```
[yatsm.algorithms.change]
CCDCesque=yatsm.algorithms.ccdc:CCDCesque
```

This entry point definition links the name of the algorithm, "CCDCesque", to the module (`yatsm.algorithms.ccdc`) containing the relevant time series algorithm class, `yatsm.algorithms.ccdc.CCDCesque`. Users select the "CCDCesque" algorithm by defining it their configuration files.

Using entry points, you can create your own algorithms, distribute them in separate Python packages, and YATSM will be able to find them and enable them to work within the YATSM package.

Behind the scenes

YATSM uses the function `iter_entry_points` from the setuptools module to find and load all entry points associated with the YATSM package.

Todo

Create example `yatsm_algorithms` repository to act as a template for including additional algorithms via setuptools.

References:

- https://pythonhosted.org/setuptools/setuptools.html
- http://stackoverflow.com/questions/774824/explain-python-entry-points
- http://stackoverflow.com/questions/13545289/how-do-i-add-a-setuptools-entry-point-as-an-example-in-my-main-project
- https://docs.python.org/3/library/pkgutil.html

CHAPTER 3

Command Line Interface Utilities

The Command Line Interface (CLI) for YATSM is built using click and is accessed using the `yatsm` command. Documentation about the CLI is available below:

Command Line Utilities

The following are command line utilities included in YATSM. When `yatsm` is installed on your system, these scripts are installed in a location in your `$PATH` and should be available at the command line.

yatsm

YATSM Command Line Interface and Subcommands:

```
$ yatsm --help
Usage: yatsm [OPTIONS] COMMAND [ARGS]...

  YATSM command line interface

Options:
  --version                Show the version and exit.
  --num_threads <threads>  Number of threads for OPENBLAS/MKL/OMP used in
                           NumPy  [default: 1]
  -v, --verbose            Be verbose
  --verbose-yatsm          Show verbose debugging messages in YATSM algorithm
  -q, --quiet              Be quiet
  -h, --help               Show this message and exit.

Commands:
  cache      Create or update cached timeseries data for YATSM
  changemap  Map change found by YATSM algorithm over time period
  classify   Classify entire images using trained algorithm
  line       Run YATSM on an entire image line by line
```

```
  map         Make map of YATSM output for a given date
  pixel       Run YATSM algorithm on individual pixels
  segment     † Warning: could not load plugin. See `build_cli_docs.py segment
              --help`.
  train       Train classifier on YATSM output
```

yatsm pixel

Run YATSM for individual pixels with specified parameters:

```
$ yatsm pixel --help
Usage: yatsm pixel [OPTIONS] <config> <px> <py>

Options:
  --band <n>            Band to plot  [default: 1]
  --plot [TS|DOY|VAL]   Plot type  [default: TS]
  --ylim <min> <max>    Y-axis limits
  --style <style>       Plot style  [default: ggplot]
  --cmap <cmap>         DOY/VAL plot colormap  [default: viridis]
  --embed               Drop to (I)Python interpreter at various points
  --seed TEXT           Set NumPy RNG seed value
  --algo_kw TEXT        Algorithm parameter overrides
  --help                Show this message and exit.
```

yatsm line

Run YATSM in batch mode for entire lines:

```
$ yatsm line --help
Usage: yatsm line [OPTIONS] <config> <job_number> <total_jobs>

Options:
  --check_cache  Check that cache file contains matching data
  --resume       Do not overwrite preexisting results
  --do-not-run   Do not run YATSM (useful for just caching data)
  --help         Show this message and exit.
```

yatsm cache

Create or modify dataset cache files used by YATSM. This command can create cache files from scratch or update existing cache files to reflect additions or removals from the time series dataset originally cached to disk.

Usage:

```
$ yatsm cache --help
Usage: yatsm cache [OPTIONS] <config> <job_number> <total_jobs>

Options:
  --update <pattern>  Create new cache files by updating old cache files
                      matching provided pattern
  --interlace         Assign rows interlaced by job instead of sequentially
  --help              Show this message and exit.
```

yatsm train

Train classifier predicting land cover for each time segment within YATSM output:

```
$ yatsm train --help
Usage: yatsm train [OPTIONS] <config> <classifier_config> <model>

  Train a classifier from ``scikit-learn`` on YATSM output and save result
  to file <model>. Dataset configuration is specified by <yatsm_config> and
  classifier and classifier parameters are specified by <classifier_config>.

Options:
  --kfold INTEGER  Number of folds in cross validation (default: 3)
  --seed INTEGER   Random number generator seed
  --plot           Show diagnostic plots
  --diagnostics    Run K-Fold diagnostics
  --overwrite      Overwrite output model file
  --help           Show this message and exit.
```

yatsm classify

Classify all lines within an image using a trained classifier:

```
$ yatsm classify --help
Usage: yatsm classify [OPTIONS] <config> <trained algorithm> <job_number>
                      <total_jobs>

Options:
  --resume  Resume classification (don't overwrite)
  --help    Show this message and exit.
```

yatsm changemap

Create maps of change information from YATSM output:

```
$ yatsm changemap --help
Usage: yatsm changemap [OPTIONS] <map_type> <start_date> <end_date> <output>

  Examples: TODO

Options:
  --root <directory>        Root timeseries directory  [default: ./]
  -r, --result <directory>  Directory of results  [default: YATSM]
  -i, --image <image>       Example timeseries image  [default: example_img]
  --date <format>           Input date format  [default: %Y-%m-%d]
  --ndv <NoDataValue>       Output NoDataValue  [default: -9999]
  -f, --format <driver>     Output format driver  [default: GTiff]
  --out_date <format>       Output date format  [default: %Y%j]
  --warn-on-empty           Warn user when reading in empty results files
  --magnitude               Add magnitude of change as extra image (pattern is
                            [name]_mag[ext])
  --help                    Show this message and exit.
```

yatsm map

Create maps of coefficients, predicted images, or land cover for any given date from YATSM output:

```
$ yatsm map --help
Usage: yatsm map [OPTIONS] <map_type> <date> <output>

  Map types: coef, predict, class, pheno

  Map QA flags:
      - 0 => no values
      - 1 => before
      - 2 => after
      - 3 => intersect

  Examples:
  > yatsm map --coef intercept --coef slope
  ... --band 3 --band 4 --band 5 --ndv -9999
  ... coef 2000-01-01 coef_map.gtif

  > yatsm map -c intercept -c slope -b 3 -b 4 -b 5 --ndv -9999
  ... coef 2000-01-01 coef_map.gtif

  > yatsm map --date "%Y-%j" predict 2000-001 prediction.gtif

  > yatsm map --result "YATSM_new" --after class 2000-01-01 LCmap.gtif

  Notes:
      - Image predictions will not use categorical information in timeseries
        models.

Options:
  --root <directory>        Root timeseries directory  [default: ./]
  -r, --result <directory>  Directory of results  [default: YATSM]
  -i, --image <image>       Example timeseries image  [default: example_img]
  --date <format>           Input date format  [default: %Y-%m-%d]
  --ndv <NoDataValue>       Output NoDataValue  [default: -9999]
  -f, --format <driver>     Output format driver  [default: GTiff]
  --warn-on-empty           Warn user when reading in empty results files
  -b, --band <band>         Bands to export for coefficient/prediction maps
  -c, --coef <coef>         Coefficients to export for coefficient maps
  --after                   Use time segment after <date> if needed for map
  --before                  Use time segment before <date> if needed for map
  --qa                      Add QA band identifying segment type
  --refit_prefix TEXT       Use coef/rmse with refit prefix for
                            coefficient/prediction maps  [default: ]
  --amplitude               Export amplitude of sin/cosine pairs instead of
                            individual coefficient estimates
  --predict-proba           Include prediction probability band (scaled by
                            10,000)
  --help                    Show this message and exit.
```

gen_date_file.sh

Generate Date and Filename Lists:

```
$ gen_date_file.sh -h

    usage: ./../scripts/gen_date_file.sh [options] <root_directory> <output_file>

    This script will output a CSV list of the date and full filepath for
    all images within <root_directory> to <output_file>. Currently the date
    must be in YYYYDOY format.

    Options:
        -p          Filename pattern [default: L*stack]
        -d          Starting index of date within filename [default: 9]
        -s          Starting index of sensor within filename [default: 0]
        -r          Use relative paths
        -o          Overwrite <output_file> if exists
        -v          Be verbose
        -h          Show help
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

References

Bibliography

[1] Zhe Zhu and Curtis E. Woodcock. Continuous change detection and classification of land cover using all available landsat data. *Remote Sensing of Environment*, 144(0):152 – 171, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0034425714000248, doi:http://dx.doi.org/10.1016/j.rse.2014.01.011.

[1] Zhe Zhu and Curtis E. Woodcock. Continuous change detection and classification of land cover using all available landsat data. *Remote Sensing of Environment*, 144(0):152 – 171, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0034425714000248, doi:http://dx.doi.org/10.1016/j.rse.2014.01.011.

Index

C