deforest Documentation

Release 1

Samuel Bowers

Jun 14, 2021

Contents

The DEnse FOREst Time Series (DEFOREST) tool is a method for detecting changes in forest cover in a time series of Sentinel-2 data.

The processing chain does the following:

- Takes Sentinel-2 data (L1C/L2A) as input.
- Trains a model to robustly identify forest/nonforest across any phenological state.
- Classifies each image to a probability of forest cover.
- Combines these images under a Bayesian framework to identify forest cover change.
- Outputs maps of forest cover change, and ‘early warnings’ of change in recent imagery.

CHAPTER 1

Contents:

1.1 Setup instructions

1.1.1 Requirements

This toolset is written for use in Linux.

You will need access to a PC or server with at least:

- Python 3
- SMFM *sen2mosaic <https://github.com/smfm-project/sen2mosaic/>*_
- 8 GB of RAM for classification

Note that processing of large volumes of Sentinel-2 data will require access substantial file storage and processing power. For user over large areas we advice use of a Linux server orsuitably equipped cloud platform.

1.1.2 Installing Anaconda Python

These tools are written in Python. We recommend the Anaconda distribution of Python, which contains all the modules necessary to run these scripts.

To install Anaconda Python, open a terminal window, change directory to the location you'd like to download Anaconda Python, and run the following commands:

```
wget https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
chmod +x Anaconda3-2019.03-Linux-x86_64.sh
./Anaconda3-2019.03-Linux-x86_64.sh
```

If this has functioned, on executing `python` in a terminal window, you should see the following:

```
Python 2.7.14 |Anaconda, Inc.| (default, Dec  7 2017, 17:05:42)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1.1.3 Setting up your Anaconda environment

Note: The Anaconda environment required for deforest can also be used for sen2mosaic. This means that sen2mosaic can be installed within the `deforest` environment without causing problems.

To ensure you are working with the appropriate version of Python as well as the correct modules, we recommend that you create an Anaconda virtual environment set up for running `deforest`. This is done by running the following commands in your terminal or the Anaconda prompt (recommended procedure):

```
conda create -n deforest -c conda-forge python=3.7 scipy pandas psutil scikit-image␣
↪scikit-learn gdal opencv pyshp
```

Activate the `deforest` environment whenever opening a new terminal window by running this command:

```
conda activate deforest
```

1.1.4 Installing sen2mosaic

If you've not already installed it, sen2mosaic can be downloaded to a machine from its repository . To do this, open a terminal window and input:

```
git clone https://github.com/smfm-project/sen2mosaic.git
```

To install sen2mosaic, navigate to the sen2mosaic directory and run the following *within your sen2mosaic environment*.

```
python setup.py install
```

1.1.5 Installing deforest

Open a terminal window and input:

```
git clone https://github.com/smfm-project/deforest.git
```

To install deforest, navigate to the deforest directory and run the following *within your deforest environment*.

```
python setup.py install
```

To avoid having to reference the full path of the Python scripts in deforest, it's a good idea add the following line to your `.bashrc` file:

```
echo "alias deforest='deforest() { python ~/deforest/cli/\"\$1\".py \$(shift; echo \"\
↪$@\") ;}; _deforest'" >> ~/.bashrc
```

Make sure you replace ~/ with the path to your installation of deforest.

1.1.6 Where do I get help?

For assistance in setting up and using deforest or sen2mosaic, email sam.bowers@ed.ac.uk.

1.2 Using deforest on the command line

1.2.1 Downloading and Preprocessing Data

Instructions for downloaded and pre-processing data from Sentinel-2 can be found at sen2mosaic.

Note: SMFM deforest is designed for analysis of dense time series of Sentinel-2 data, which will require access to subsantial file storage and processing power. We recommend use of a cloud platform, where data do not have to downloaded of pre-processed on a local machine. See, for example, the 'Data and Information Access Services' (DIAS) platforms that provide centralised access to Copernicus data.

1.2.2 Calibrating SMFM deforest

SMFM deforest will work best where local data are used for calibration. This is performed in two steps: (i) Extracting training data, and (ii) Training the classifier.

Extracting training data

Training pixels are defined by either a geotiff image or shapefile defining locations of stable forest and stable non-forest. Classification features are derived for a series of Sentinel-2 images, and a random selection of pixel values extracted for forest and non-forest classes. The output is an array of feature values used to train a classifier to predict probabilities of forest and non-forest.

```
usage: extract.py [-h] [-te XMIN YMIN XMAX YMAX] [-e EPSG] [-res N]
                  [-t SHP/TIF] [-f [VALS [VALS ...]]] [-nf [VALS [VALS ...]]]
                  [-fn NAME] [-l 1C/2A] [-mi N] [-mp N] [-o DIR] [-n NAME]
                  [-p N] [-v]
                  [FILES [FILES ...]]

Extract indices from Sentinel-2 data to train a classifier of forest cover.
Returns a numpy .npz file containing pixel values for forest/nonforest.

positional arguments:
  FILES                 Sentinel 2 input files (level 2A) in .SAFE format.
                        Specify one or more valid Sentinel-2 .SAFE, a
                        directory containing .SAFE files, multiple tiles
                        through wildcards (e.g. *.SAFE/GRANULE/*), or a text
                        file listing files. Defaults to processing all tiles
                        in current working directory.

required arguments:
  -te XMIN YMIN XMAX YMAX, --target_extent XMIN YMIN XMAX YMAX
                        Extent of output image tile, in format <xmin, ymin,
                        xmax, ymax>.
  -e EPSG, --epsg EPSG  EPSG code for output image tile CRS. This must be UTM.
                        Find the EPSG code of your output CRS as
```

(continues on next page)

(continued from previous page)

```
                        https://www.epsg-registry.org/.
  -res N, --resolution N
                        Specify a resolution to output.
  -t SHP/TIF, --training_data SHP/TIF
                        Path to training data geotiff/shapefile.
  -f [VALS [VALS ...]], --forest_values [VALS [VALS ...]]
                        Values indicating forest in the training GeoTiff or
                        shapefile
  -nf [VALS [VALS ...]], --nonforest_values [VALS [VALS ...]]
                        Values indicating nonforest in the training GeoTiff or
                        shapefile

optional arguments:
  -fn NAME, --field_name NAME
                          Shapefile attribute name to search for training data
                          polygons. Defaults to all polygons. Required where
                          inputting a shapefile as training_data.
  -l 1C/2A, --level 1C/2A
                          Input image processing level, '1C' or '2A'. Defaults
                          to '2A'.
  -mi N, --max_images N
                          Maximum number of input tiles to extract data from.
                          Defaults to all valid tiles.
  -mp N, --max_pixels N
                          Maximum number of pixels to extract from each image
                          per class. Defaults to 5000.
  -o DIR, --output_dir DIR
                          Output directory. Defaults to current working
                          directory.
  -n NAME, --output_name NAME
                          Specify a string to precede output filename. Defaults
                          to 'S2'.
  -p N, --n_processes N
                          Maximum number of tiles to process in paralell. Bear
                          in mind that more processes will require more memory.
                          Defaults to 1.
  -v, --verbose         Make script verbose.
```

For example, for a directory containing Sentinel-2 data from tile `36KWD` (`~/S2_data/`), specifying an appropriate bounding box and resolution (`-r`, `-e`, `-te`), training data contained in a geotiff (`~/training_data.tif`) coded as stable forest (`-f 1`) and stable non-forest (`-nf 2`), using 10 processes (`-p 10`):

```
deforest extract ~/S2_data/ -r 20 -e 32736 -te 399980 7790200 609780 7900000 -t ~/
↪training_data.tif --max_images 100 -f 1 -nf 2 -v -p 10
```

Training the model

SMFM deforest uses a Random Forest model to predict the probability of forest in each input Sentinel-2 image. This model can be calibrated using training data from the region of interest.

The training function takes a series of labelled forest and non-forest pixels (see 'Extracting training data') as input and returns a calibrated model (a `.pkl` file). The process also returns a series of plots that can b eused to assess model performance.

```
usage: train.py [-h] [-m N] [-n NAME] [-o PATH] DATA

Ingest Sentinel-2 data to train a random forest model to predict the
probability of a pixel being forested. Returns a calibrated model and QA
graphics.

positional arguments:
  DATA                  Path to .npz file containing training data, generated
                        by extract.py

optional arguments:
  -m N, --max_samples N
                        Maximum number of samples to train the classifier
                        with. Smaller sample sizes will run faster and produce
                        a simpler model, possibly at the cost of predictive
                        power. Defaults to 100,000 points.
  -n NAME, --output_name NAME
                        Specify a string to precede output filename. Defaults
                        to name of input training data.
  -o PATH, --output_dir PATH
                        Directory to save the classifier. Defaults to the
                        current working directory.
```

For example, using the output of `deforest extract`:

```
deforest train S2_training_data.npz
```

1.2.3 Classification and change detection

SMFM deforest uses a two-step process to produce change maps: (i) classification of individual Sentinel-2 images, and (ii) change detection.

Image classification

Sentinel-2 images are classified into a continuous probability of forest in each non-masked pixel. Inputs can be either Sentinel-1 L1C data or L2A data (preferable). The output is a set of geotiffs numbered 0 - 100%, with a set extent, resolution and coordinate reference system (UTM).

```
usage: classify.py [-h] [-te XMIN YMIN XMAX YMAX] [-e EPSG] [-r N] [-m PKL]
                   [-l 1C/2A] [-p N] [-n NAME] [-o DIR]
                   [FILES [FILES ...]]

Process Sentinel-2 to match a predefined CRS and classify each to show a
probability of forest (0-100%) in each pixel.

required arguments:
  -te XMIN YMIN XMAX YMAX, --target_extent XMIN YMIN XMAX YMAX
                        Extent of output image tile, in format <xmin, ymin,
                        xmax, ymax>.
  -e EPSG, --epsg EPSG  EPSG code for output image tile CRS. This must be UTM.
                        Find the EPSG code of your output CRS as
                        https://www.epsg-registry.org/.
  -r N, --resolution N  Specify a resolution to output.
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  FILES                 Sentinel 2 input files in .SAFE format. Specify one or
                        more valid Sentinel-2 .SAFE files, a directory
                        containing .SAFE files, or multiple granules through
                        wildcards (e.g. *.SAFE/GRANULE/*). Defaults to
                        processing all granules in current working directory.
  -m PKL, --model PKL   Path to .pkl model, produced with train.py. Defaults
                        to a test model, trained on data from Chimanimani in
                        Mozambique.
  -l 1C/2A, --level 1C/2A
                        Processing level to use, either '1C' or '2A'. Defaults
                        to level 2A.
  -p N, --n_processes N
                        Maximum number of tiles to process in paralell. Bear
                        in mind that more processes will require more memory.
                        Defaults to 1.
  -n NAME, --output_name NAME
                        Specify a string to precede output filename. Defaults
                        to 'S2'.
  -o DIR, --output_dir DIR
                        Optionally specify an output directory
```

For example, to classify probability of forest in all images in a directory containing Sentinel-2 data from tile `36KWD` (`~/S2_data/`), specifying an appropriate bounding box and resolution (`-r`, `-e`, `-te`), and a calibrated model named `S2_model.pkl`:

```
deforest classify ~/S2_data/ -r 20 -e 32736 -te 399980 7790200 609780 7900000 -m S2_
↪model.pkl
```

Change detection

The final step is to combine the time series of forest probability images under a Bayesian framework to detect changes in forest cover. The output is two geotiffs, one providing the year of change, the other an early warning of pixels flagged as possible changes at the final time step.

```
usage: change.py [-h] [-t N] [-b N] [-o DIR] [-n NAME] FILES [FILES ...]

Process probability maps to generate a map of deforestation year and warning
estimates of upcoming events.

required arguments:
  FILES                 A list of files output by classify.py, specifying
                        multiple files using wildcards.

optional arguments:
  -t N, --threshold N   Set a threshold probability to identify deforestation
                        (between 0 and 1). High thresholds are more strict in
                        the identification of deforestation. Defaults to 0.99.
  -b N, --block_weight N
                        Set a block weighting threshold to limit the range of
                        forest/nonforest probabilities. Set to 0 for no block-
                        weighting. Parameter cannot be set higher than 0.5.
  -o DIR, --output_dir DIR
                        Optionally specify an output directory. If nothing
                        specified, downloads will output to the present
```

(continues on next page)

(continued from previous page)

```
                        working directory, given a standard filename.
  -n NAME, --output_name NAME
                        Optionally specify a string to precede output
                        filename. Defaults to the same as input files.
```

For example, using default change detection parameters and a set of classified images from `classify.py`:

```
deforest change ./*.tif
```

1.3 Worked example on the command line

Here will be shown by example how the `deforest` processing chain works in practice. We will focus on an example from Manica and Manicaland provinces of Mozambique and Zimbabwe, with the aim of producing a remote sensing product for annual deforestation and near real-time warnings of tree cover change.

The study area for this example straddles the border of Mozambique and Zimbabwe, and is appropriate as it is relatively densely forested, with high rates of forest cover change. We'll use a dense time series of two Sentinel-2 tiles, **36KVD** and **36KWD**. This location has the CRS **UTM 36S** (EPSG: 32736), and an extent of **400,000 - 600,000** m Eastings, and **7,800,000 - 7,900,000** m Northings. This example will use all data from the start of the Sentinel-2 era to mid-July 2019, the time of writing.

1.3.1 Preparation

First insure that the setup instructions have been completed successfully.

Open a terminal, and use `cd` to navigate to the location you'd like to store data.

```
[user@linuxpc ~]$ mkdir worked_example
[user@linuxpc ~]$ cd worked_example
```

Use mkdir to make a separate directory to contain the data you wish to download.

```
[user@linuxpc worked_example]$ mkdir DATA
```

To begin, navigate to the DATA folder.

```
[user@linuxpc worked_example]$ cd DATA
```

1.3.2 Data preparation

Downloading data

The first step is to download Sentinel-2 level 1C data from the Copernicus Open Access Data Hub.

For this we use the `sen2mosaic download.py` tool, installed as part of setup for `deforest`. See sen2mosaic documentation for more details.

Here, the process will be to download all L2A data for the period 1st June 2018 to 2019 for the tile `36KWD` specifying a maximum cloud cover percetage of 30%:

```
[user@linuxpc DATA]$ s2m download -u user.name -p supersecret -t 36KWD -c 30 -s␣
↪20180601 -l 2A
```

Note: Not all Sentinel-2 data are available in L2A format, meaning that the user can either use L1C data, or preprocess data with sen2cor. See sen2mosaic for more details.

deforest works using long data time series, and will perform poorly where a time series is too short. Try to ensure access to at least 3 years worth of data. The remainder of this worked example will assume access to every L2A image from the start of the Sentinel-2 era to July 2019.

Note: Data from more than 1 year in the past may have been moved to the Long Term Archive. To access this data it's necessary to order it from Copernicus, which can be a laborious task. For most practical purposes, to use dense time series you should consider running these scripts on an online platform (e.g. F-TEP, DIAS, AWS) where data are stored locally to processing infrastructure.

Before contintuing, ensure that you have a directory (i.e. DATA) containing a series of Sentinel-2 .SAFE files. In this case, there will be data for two Sentinel-2 tiles, 36KVD and 36KWD:

```
[user@linuxpc DATA]$ ls
S2B_MSIL2A_20170630T072949_N0205_R049_T36KWC_20170630T075509.SAFE
S2B_MSIL2A_20170710T072619_N0205_R049_T36KWC_20170710T074330.SAFE
S2B_MSIL2A_20170713T075209_N0205_R092_T36KVD_20170713T075751.SAFE
S2B_MSIL2A_20170713T075209_N0205_R092_T36KWC_20170713T075751.SAFE
S2B_MSIL2A_20170713T075209_N0205_R092_T36KWC_20170713T080544.SAFE
S2B_MSIL2A_20170713T075209_N0205_R092_T36KWD_20170713T075751.SAFE
S2B_MSIL2A_20170720T074239_N0205_R049_T36KWC_20170720T074942.SAFE
S2B_MSIL2A_20170723T073609_N0205_R092_T36KVD_20170723T075425.SAFE
...
S2B_MSIL2A_20190703T073619_N0212_R092_T36KWD_20190703T122423.SAFE
S2B_MSIL2A_20190713T073619_N0213_R092_T36KWD_20190713T111309.SAFE
S2B_MSIL2A_20190723T073619_N0213_R092_T36KWD_20190723T115930.SAFE
```

1.3.3 Training the classifier

Training of the classifier is performed in two steps: i) Extracting data from a series of training pixels of stable forest and nonforest, and ii) Calibrating a classifier to separate the spectral characteristics of forest from those of nonforest.

Note: Both extraction and calibration steps *may* be skipped for the case of tiles 36KVD and 36KWD, as deforest is provided with a default classifier trained at this location. For all other locations, it's strongly recommended that these steps are followed.

Extracting training data

The first step to using the deforest algorithm is to extract training data. This task is performed with the deforest extract.py tool.

There are two options for specification of locations to extract training data, either using a shapefile or a raster image. In each case we need to specify the attributes of a 'forest' and a 'nonforest' pixel, and these should be associated with locations of stable forest/nonforest that do not change class over the course of the training period.

For ease, here we'll use a pre-existing land cover map to train our classifier (download on registration here). This map covers Africa at 20 m resolution for the year 2016, and it will be assumed that these classes are accurate and do not change between 2016 - 2019. This map has numbered land cover classes with meaning:

Land cover	Value
No data	0
Tree cover areas	1
Shrubs cover areas	2
Grassland	3
Cropland	4
Vegetation aquatic or regularly flooded	5
Lichens Mosses / Sparse vegetation	6
Bare areas	7
Built up areas	8
Snow and/or ice	9
Open water	10

To apply this to data in the existing directory containing Sentinel-2 data, use the following command:

```
[user@linuxpc worked_example]$ deforest extract path/to/DATA/ -r 20 -e 32736 -te␣
↪399980 7790200 609780 7900000 -t path/to/ESACCI-LC-L4-LC10-Map-20m-P1Y-2016-v1.0.
↪tif -f 1 -nf 2 3 4 5 6 7 8 10 -v
```

This translates to extracting features from a random subset of forest (`-f`) and nonforest (`-nf`) pixels from a geotiff image (`-t`) on in each image contained within `path/to/DATA`, with a specified resolution (`-r`), extent (`-e`) and a coordinate reference system specified by the EPSG code (`-e`).

If computational resources are limited, input training data can be limited to fewer images:

```
[user@linuxpc worked_example]$ deforest extract path/to/DATA/ -r 20 -e 32736 -te␣
↪399980 7790200 609780 7900000 -t path/to/ESACCI-LC-L4-LC10-Map-20m-P1Y-2016-v1.0.
↪tif -o ./ --max_images 100 -f 1 -nf 2 3 4 5 6 7 8 10 -v
```

And if more computational resources are available, this process can be sped up by increasing the number of processes to, for instance, to run 8 similtaneous processes:

```
[user@linuxpc worked_example]$ deforest extract path/to/DATA/ -r 20 -e 32736 -te␣
↪399980 7790200 609780 7900000 -t path/to/ESACCI-LC-L4-LC10-Map-20m-P1Y-2016-v1.0.
↪tif -o ./ -f 1 -nf 2 3 4 5 6 7 8 10 -v -p 8
```

Be aware, the more processes used the more computational resources will be required.

The user also specify a larger number of pixels to extract from each image (default: 5000 per class) using the `--max_pixels` (`-mp`) option:

```
[user@linuxpc worked_example]$ deforest extract path/to/DATA/ -r 20 -e 32736 -te␣
↪399980 7790200 609780 7900000 -t path/to/ESACCI-LC-L4-LC10-Map-20m-P1Y-2016-v1.0.
↪tif -f 1 -nf 2 3 4 5 6 7 8 10 -mp 10000 -v
```

The output of this command will be a `.npz` file, which contains the pixel values for each classification feature.

```
[user@linuxpc worked_example]$ ls
S2_training_data.npz
...
```

Calibrating the classifier

The next step is to use this training data to calibrate the classifier of forest cover. This is performed with the `deforest train.py` tool. This tool takes the feature values from `extract.py` to train a classifier of forest/nonforest based on those feature values.

To train the classifier, run the command:

```
[user@linuxpc worked_example]$ deforest train S2_training_data.npz
```

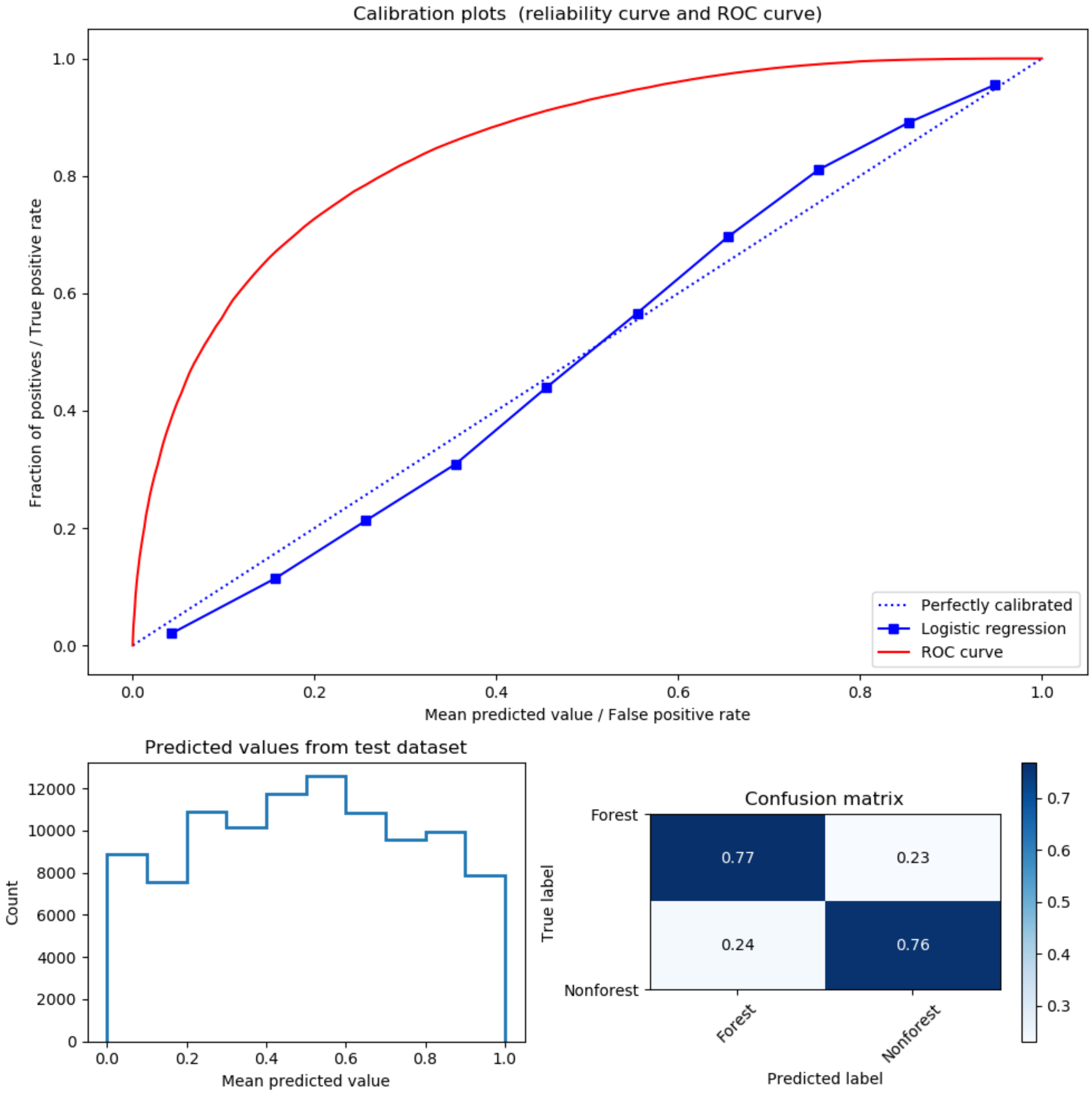
To increase the complexity of the classifier, a larger number of pixel values from be subsampled from the input data (default = 100,000 pixels). For example, to train the model based on 200,000 pixels:

```
[user@linuxpc worked_example]$ deforest train S2_training_data.npz -m 200000
```

Once complete there will be two new files:

```
[user@linuxpc worked_example]$ ls
S2_model.pkl
S2_quality_assessment.png
...
```

`S2_model.pkl` is an object that defines the calibrated classifier, `S2_quality_assessment.png` can be used to assess the quality of the model:



This figure shows three plots to aid interpretation of model quality:

- **Calibration plots** The plot at the top shows a calibration curve in blue and a receiver operating characteristic (ROC) curve. The closer the calibration curve is to the 1:1 line, and the further to the upper left the ROC curve reaches, the higher the quality of the calibration.
- **Predicted values** The plot at the bottom-left shows the distribution of forest probability predictions (0 to 1) from a random sample of the training data. Where a classifier is highly confident in predicting forest cover there will be peaks close to 0 (non-forest) and 1 (forest), and where less certain (such as this case) the distrubution will be flatter. To be avoided are classifiers that rarely confidently predict forest or non-forest.
- **Confusion matrix** The plot at the bottom-right is a confusion matrix, where input data labels (forest / non-forest) are compared to classifier predictions (< 50% = non-forest, > 50% = forest). A good classifier will generally score highly in the top-left and bottom-right boxes (true postives/negatives) and low in the bottom-left and top-right boxes (false positives/negatives).

The user should explore calibrating the classifier with multiple options until classifier results are satisfactory. An advanced user may choose to alter the input features for the classifier through modification of the `loadFeatures()` function in `deforest/classify.py`.

1.3.4 Classifing the data

First, the user should make a new directory to store classified images:

```
[user@linuxpc worked_example]$ mkdir classified_images
```

The user can then run the classification algorithm that was just calibrated to produce probability of forest for each image. This operates very similarly to `training.py`, here using the same output extents:

```
[user@linuxpc worked_example]$ deforest classify path/to/DATA/ -m S2_model.pkl -r 20 -
↪e 32736 -te 399980 7790200 609780 7900000 -o classified_images/
```

This translates to classifying images contained within `path/to/DATA` using a trained classifier (`-m`), with a specified output resolution (`-r`), extent (`-te`) and projection (`-e`), and outputing classified images to a directory (`-o`).

Note: If using the default model in place of a locally calibrated model, omit the `-m S2_model.pkl` option to use the default model.

If resources are available, classification can can be sped up by allocating additional processes:

```
[user@linuxpc worked_example]$ deforest classify path/to/DATA/ -m S2_model.pkl -r 20 -
↪e 32736 -te 399980 7790200 609780 7900000 -o classified_images -p 8
```

If without access to preprocessed L2A Sentinel-2 images, L1C data can be input using the `-l` option:

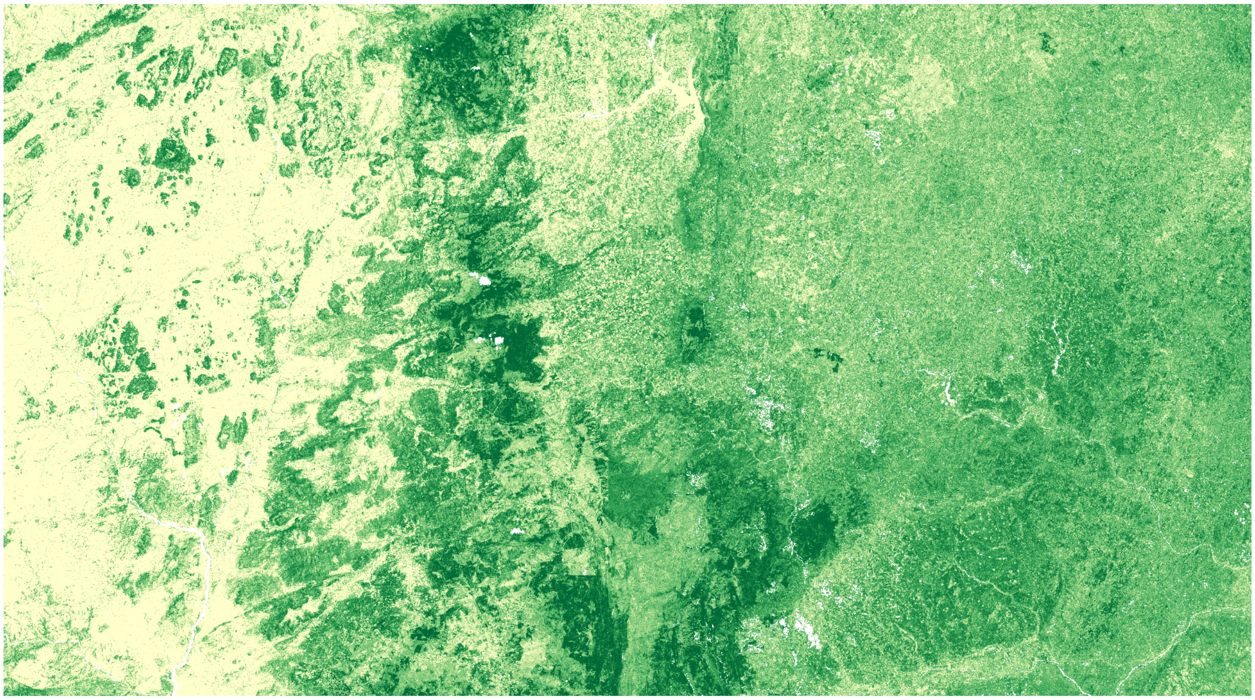
```
[user@linuxpc worked_example]$ deforest classify path/to/DATA/ -m S2_model.pkl -r 20 -
↪e 32736 -te 399980 7790200 609780 7900000 -o classified_images -l 1C
```

Once complete, images will be output to the `classified_images` directory:

```
[user@linuxpc classified_images]$ ls
S2_S2_T36KVD_20151126_075714.tif  S2_S2_T36KWC_20171001_075742.tif
S2_S2_T36KVD_20151206_075547.tif  S2_S2_T36KWC_20171006_075832.tif
S2_S2_T36KVD_20151226_080933.tif  S2_S2_T36KWC_20171008_075024.tif
S2_S2_T36KVD_20151229_082023.tif  S2_S2_T36KWC_20171016_075320.tif
S2_S2_T36KVD_20160105_080719.tif  S2_S2_T36KWC_20171023_074855.tif
S2_S2_T36KVD_20160108_082023.tif  S2_S2_T36KWC_20171026_080348.tif
S2_S2_T36KVD_20160125_080606.tif  S2_S2_T36KWC_20171031_075502.tif
S2_S2_T36KVD_20160204_080212.tif  S2_S2_T36KWC_20171107_075205.tif
S2_S2_T36KVD_20160207_080537.tif  S2_S2_T36KWC_20171120_075322.tif
...                               ...
S2_S2_T36KWC_20170926_075507.tif  S2_S2_T36KWD_20180906_075434.tif
S2_S2_T36KWC_20170928_074401.tif
```

These images will each have pixels numbered 0 to 100%, representing the probability of forest at that time point. The nodata value is 255.

For instance, this image shows forest probability in the study region for two images (`36KVD`: 01/10/2016, `36KWD`: 26/11/2015), with pixels shown in darker green indicating a higher probability of forest in that image.



1.3.5 Change detection

The final step is to combine these classified images into an estimate of forest cover and forest cover change. For use the `change.py` command line tool:

```
[user@linuxpc worked_example]$ deforest change classified_images/*.tif
```

This process will combine all the probability images in `classified_images`, and identify changes in the time series.

Options are available to change the parameters of change detection. For example, to apply a stricter probability threshold detection of confirmed changes:

```
[user@linuxpc worked_example]$ deforest change -t 0.9995 classified_images/*.tif
```

Or to alter block-weighting, which reduces the impact of very high or low probability outliers:

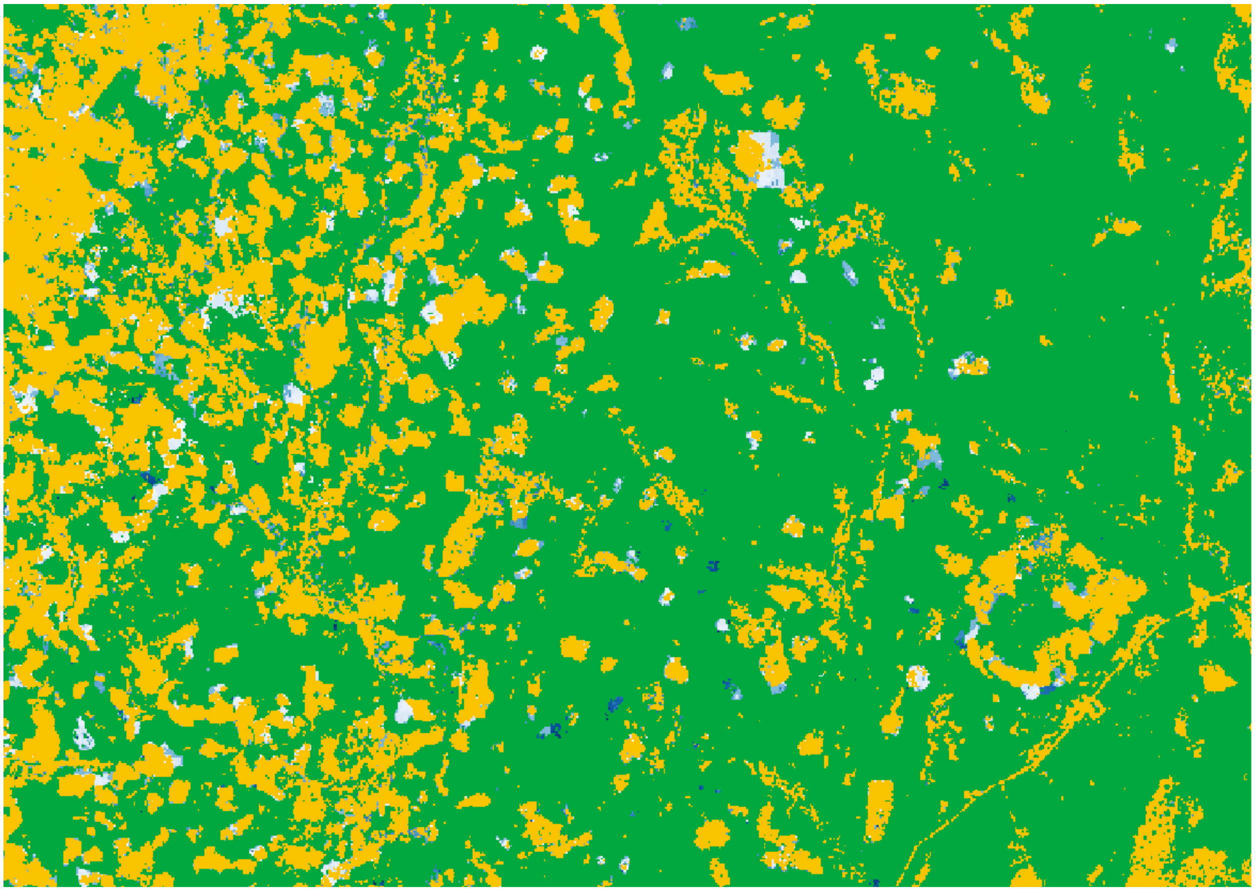
```
[user@linuxpc worked_example]$ deforest change -b 0.2 classified_images/*.tif
```

The `change.py` script will output two images:

```
[user@linuxpc worked_example]$ ls
...
S2_confirmed.tif
S2_warning.tif
```

The image `S2_confirmed.tif` shows the year of changes that have been detected in the time series (in this case 2017-2019.5). Locations of remaining forest are numbered 0. Note that the changes in the first year or two will be mostly spurious, as the landscape is initially considered entirely forested. It is recommended that the user either discards the first 2-3 years of changes, or uses a high-quality forest baseline map to mask out locations that weren't forest at the start of the time series.

Shown below is a section of `S2_confirmed.tif`. Forest is shown in green, and non-forest in yellow (change date < 2018.5). Confirmed changes (2018.5 - 2019.5) are indicated in blue, with lighter blues changing earlier in the time series.



The image `S2_warning.tif` shows the combined probability of non-forest existing at the end of the time series in locations that have not yet been flagged as deforested. This can be used to provide information on locations that have not yet reached the threshold for confirmed changes, but are looking likely to possible. A simple probability threshold can be applied to supply early warnings.

Shown below is a section of `S2_warning.tif`, with warning locations where probability of non-forest is greater than 85% shown in red. Early warnings show pixels that have yet to be confirmed as change, at the cost of an increased false positive rate relative to confirmed changes.