July 19, 2022

Shallow Groundwater Estimation (SAGE) Tool

Technical Methods Document

Leah Campbell
leahs.campbell@gmail.com

Table of Contents

1 Introduction

This technical methods document outlines the steps needed to understand and replicate the Shallow Groundwater Estimation Tool (SAGE) described in Rohde et al. (2021). It is designed to be used to understand and run the code included in the SAGE Github repository (https://github.com/tnc-ca-geo/SAGE.git). The workflow requires an intermediate working knowledge of Python and the Python Google Earth Engine API: it is beyond the scope of this document to assist the user to install Python or the necessary Python libraries. The scripts used to run this workflow are organized into steps, some of which can be run basically out of the box (i.e., scripts 1-3). Others require significant effort on the part of the user to collect and format the appropriate datasets. In other words, this is not a plug-and-play set of code, but rather a framework that can assist you to run this workflow in a specific area of interest using your own, tailored datasets.

The SAGE workflow uses a Random Forest model to predict depth to groundwater levels in Groundwater Dependent Ecosystems (GDEs). The predictor layers, used to train and then apply the model, include static layers (like biome or hydroregion), as well as annual climate and spectral reflectance data that has been smoothed using the LandTrendr temporal segmentation algorithm, described below.

The ability of any Random Forest model to succeed depends upon the quality of the data that is input to the model. The SAGE code streamlines the creation of Landsat and Daymet annual composites, the running of LandTrendr, and the application of the Random Forest model. However, the user is responsible for obtaining a high-quality dataset of GDE polygons and intersecting them with surrounding groundwater well data. Some of the categorical predictor layers used for the California implementation of SAGE (such as HUC 8, biome, and ecoregion) are easily obtained for any location in the United States and are therefore written into the SAGE code as default predictor layers. Others (such as vegetation macrogroup) are taken from a curated dataset specific for California. The user will need to use their own local, expert knowledge to identify appropriate categorical predictor layers that can be used for their own tailored version of this workflow. In other words, this is not a plug-and-play set of code. It requires some coding knowledge as well as work on the front end to obtain appropriate input data.

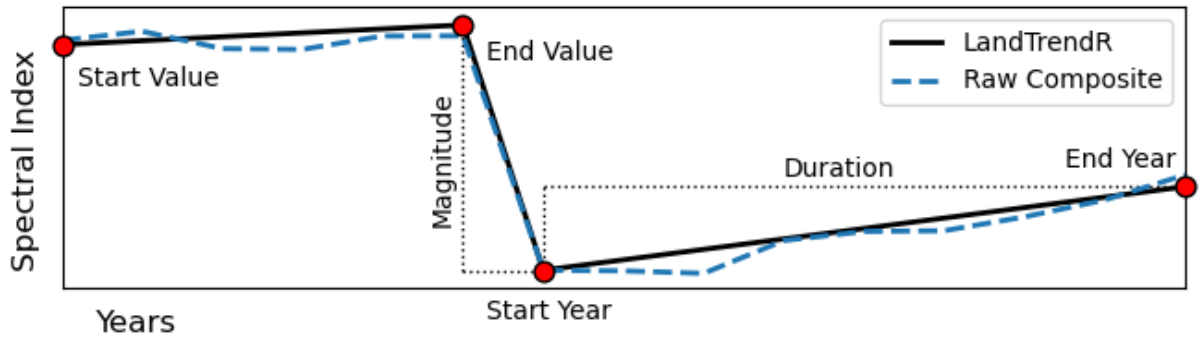

Figure 1. Example LandTrendr time series. Adapted from Figure 2.2 in the LT-GEE Guide https://emapr.github.io/LT-GEE/landtrendr.html.

A more easily streamlined part of the workflow is the creation of annual time series of Landsat surface reflectance image composites and climate composites from the Daily Surface Weather Data (Daymet) dataset (Thornton et al., 2016). The SAGE scripts will create these composites over a specified area of

interest and then run them through the LandTrendr temporal segmentation algorithm (Kennedy et al. 2010, 2018). LandTrendr runs iterative linear regressions over time series data to remove noise and partition the time series into discrete, linear segments. It was created specifically for vegetation change detection purposes and is useful for many time series applications: it is used here not only for its smoothing and denoising abilities but also to detect trends. An example of LandTrendr is shown in Figure 1 and illustrates how a time series of some spectral index is partitioned into three segments. Each segment then has a magnitude of change, a duration, and a fitted value (the value for a specific year along the black line). For more information about LandTrendr, please see the LT-GEE Guide at https://emapr.github.io/LT-GEE/landtrendr.html.

We refer the reader to Rohde et al. (2021) for additional information about the datasets and algorithms used. The remainder of this document describes the required software and datasets needed to run this code, and then steps through each of the scripts that create the SAGE workflow.

2 Set-up

2.1 Google Earth Engine

This workflow relies heavily on Google Earth Engine (GEE) to access data and run computationally intensive processes. To begin, you will need to set up a GEE account using an existing or new Gmail account. The best way to do this as of June 2022 is to follow this link, scroll down to the bottom of the page, and click "Sign Up Now." It may take a few days for your request to approved. Note that GEE is currently free for academic and research use but requires a commercial account for businesses.

2.2 Python Environment

We use the GEE Python API exclusively for our GEE processes. Therefore, you will need a working Python environment. Next, you'll need to install and authenticate the GEE Python API following the instructions here. Finally, the SAGE workflow requires the geeViz package, which can be installed by running "pip install geeViz" (Note: this SAGE version is written using geeViz version 2022.6.1).

3 Required Datasets

3.1 Groundwater Data

Annual depth-to-groundwater (DTG) data from a relevant source (i.e., USGS National Water Information System, state water database, etc.). The SAGE DTG dataset was downloaded from the California Department of Water Resources SGMA Data Viewer. This data will be used to create the trainingGDECollection described in section 3.2.

We assume that the following steps have been taken with the well data:

1. Depth to groundwater values are positive, to reflect the absolute value of the depth to groundwater below the surface.
2. Null depth-to-groundwater values have been assigned a consistent null value.
3. Wells have been filtered to only those with at least five years of observations throughout the data record (or whatever minimum year threshold you want to apply).

Later in the modeling process, you will be given the opportunity to filter for type of well (e.g., shallow only) and for well observations within a given range of acceptable values.

3.2 GDE Shapefiles

Shapefiles delineating the extent of GDEs. The geometry of each GDE should be a polygon (i.e., no points or lines). Each polygon should include information about the vegetation type. The SAGE study used only GDEs dominated by phreatophytes, which are groundwater-dependent plants that can be used as indicators of groundwater. You will want to do some pre-processing to filter for these options ahead of time, and then save your full collection of GDEs as an asset in the GEE repository.

1. applyGDECollection

   This is your full (curated) collection of GDEs that you want to apply the model on. You may want to have any special static predictor layers or strata (described in Section 3.3) included as attributes in these GDEs, although there is another opportunity to add strata in 4_ApplyTableExporter.py. The GDE collection used for Rohde et al. (2021) is from the Natural Communities Commonly Associated with Groundwater dataset described in Klausmeyer et al. (2018).

   The minimum required attributes for each of these features is:

   1. Shape_Area: or some other string, indicates the area of the polygon in whatever unit you prefer. We use m^2, but it doesn't matter as long as you know what the unit is and can identify a minimum area in that unit that you want to limit your collection to. Polygons in the applyGDECollection can be any size. Polygons in the trainingGDECollection should be at least 900 m^2 to match the minimum mapping unit of the Landsat imagery (30 m x 30 m pixels). Users can also define the minimum area of training polygons in SAGE_Initialize.py.
   2. POLYGON_ID: or some other string, an ID number for this GDE.
   3. Any other attributes you might consider using as predictors in the model (for us, Macrogroup). Note that there is a place to add predictors to your GDEs later in the workflow as well.

2. trainingGDECollection

   This is a subset of applyGDECollection, with DTG well data assigned. This collection is created by doing a spatial join between the wells (section 3.1) and the GDEs (applyGDECollection, described above). For SAGE, we require that wells be within 1 km of a GDE to be included and we filter out any that do not meet that requirement during this step. Note that since there can be many wells around one GDE polygon, a separate copy of the GDE should be made for each well-GDE combination.

   We assume that the following steps have been taken with this dataset:

   1. Depth to groundwater values are positive, to reflect the absolute value of the depth to groundwater below the surface.
   2. Null depth-to-groundwater values have been assigned a consistent null value.
   3. There is an attribute for each year of depth to groundwater data (e.g., 'Depth1986', 'Depth1987', etc.), so all years of depth to groundwater data are included as attributes in a single GDE/well combination feature. Null values for a specific year should still have a depth attribute (i.e., every feature should have the exact same number of attributes, even if some years are null).

4. GDEs have been filtered to only those within 1 km (or whatever distance you decide) of a well
5. Wells have been filtered to only those with at least five years of observations throughout the data record (or whatever minimum year threshold you want to apply).

The minimum required attributes for each of these features (GDE/well combination) are:

1. Shape_Area: or some other string, indicates the area of the polygon in whatever unit you prefer (we use m^2). Same as in apply GDE collection.
2. POLYGON_ID: or some other string, a unique ID number for this GDE. Same as in apply GDE collection.
3. Any other attributes you might consider using as predictors in the model (for us, Macrogroup). Note that there is a place to add predictors to your GDEs later in the workflow as well. Same as in GDE collection.
4. Depth_Str: or some other string, indicates the type of well. We use this to filter to only include "Shallow: perf." wells. Alternatively, you could have a constant "well depth" attribute that signals the constant depth of the well.
5. STN_ID: or some other string, a unique ID number for this well.
6. Depth1985, Depth1986, etc.: Annual depth to groundwater values. There should be an attribute for each year, even if there is no data (define a consistent no data value, e.g., -999).

3.3 Categorical Predictor Variables

The California SAGE study used HUC 8 watershed boundaries, biome, ecoregion, hydrological regions, and vegetation macrogroup as categorical predictor variables. Users of this workflow are not limited to this list and are encouraged to think of any locally relevant datasets that may influence groundwater levels (or how groundwater levels would manifest in a GDE vegetation community). Some of the predictor layers used for the California study, such as the hydrological regions, are California-specific datasets and there may not be an exact equivalent in every state. The user is encouraged to take advantage of their own expertise and local knowledge to modify this list as necessary.

For the SAGE study, these attributes (except for vegetation macrogroup, which was already part of the GDE dataset for the California study) are added to the GDE polygons using the addStrata() function in 4_ApplyTableExporter.py. Only the HUC 8 attribute is available throughout the US, the rest of the strata will need to be added by the user.

1. HUC 8 boundaries
    a. Available on Google Earth Engine (USGS/WBD/2017/HUC08) and already integrated into the SAGE script
2. Biome and Ecoregion
    a. Taken from TNC's terrestrial ecoregions of the world dataset, which can be downloaded from https://geospatial.tnc.org/datasets/b1636d640ede4d6ca8f5e369f2dc368b/about.
3. Hydrological regions

    a. Downloaded from the California Natural Resources Agency at https://gis.data.cnra.ca.gov/datasets/2a572a181e094020bdaeb5203162de15/explore?location=36.272574%2C-119.270000%2C6.74
4. Vegetation macrogroup
    a. Vegetation data grouped using the California Department of Fish and Wildlife hierarchical natural communities classification [California Department of Fish and Wildlife (CDFW), 2021]

4 Running the Code

There are eight scripts that are used to run the SAGE workflow, which is hosted on Github (https://github.com/tnc-ca-geo/SAGE.git). Each has a number in the beginning of its name indicating the order in which they should be run. Each script must be run completely and allow for any exports to finish processing in the order that they are listed. A ninth script, SAGE_Initialize.py, is automatically run as initialization for each of the other eight scripts and is the location where any user inputs and options are stored. Ideally, you should only need to modify SAGE_Initialize.py, although you will probably need to do some troubleshooting in the applyTableExporter.py and trainingTableExporter.py scripts to account for any strata/predictor layers that are unique to your study area. Each script is briefly described here, including a brief explanation of the processes that it accomplishes, as well as any input or output generated by each script.

Note that if you plan to run this workflow for the full Landsat record (1984 to present), you will run out of space in your personal asset repository, which is limited to 250 GB. You will need to request additional space for your cloud project from Google Earth Engine.

4.1 SAGE_Initialize.py

This contains all the user options, as well as some functions that are frequently used. It is divided into several sections, which are briefly described below.

4.1.1 GEE Credentials

If you have more than one GEE credential available to you, you can split up some of the more exporting-intensive processes between credentials. Each of your credentials must have write access to your root repository, and they should all be stored in the default credentials folder on your computer. This folder can be found by running the ee.oauth.get_credentials_path() command. If you only have one credential available (or if you're confused by this), you can disregard this section – just make sure that any variable with "UseMultiCredentials" in the name is set to False.

4.1.2 Global: Study Area, Years, and CRS/Transform/Scale

This section defines some global variables.

1. Study area can be defined as simply a state name, or you can use a GEE feature or geometry of your choosing.
2. CRS, transform, and scale are used for exporting raster data and the default options are optimized for the continental United States.
3. StartTrainingYear and endTrainingYear refer to the start and end years of your DTG well data.
4. StartApplyYear and endApplyYear are the start and end years that you want to run the model.

4.1.3 Global: Paths and Naming

The paths and naming conventions for the raster data and tables you will be creating and exporting, and then reading in subsequent scripts, are all defined here.

4.1.4 Predictor Layer Options

This section contains options for creating the predictor raster layers from the Landsat spectral reflectance and Daymet climate data archives. Each option is explained using comments in the script. You will make annual composites of Landsat data using 1_GetLandsatWrapper.py (Section 4.2) and annual composites of Daymet data using 2_GetClimateWrapper.py (Section 4.3). Then, you will run LandTrendr on both collections of annual composites (Section 4.4).

4.1.5 Training and Apply Data Options

Filtering options for GDE/well data are included here. This may require special attention to make sure that the format of your GDE feature collection matches the format used here. Then, there are just a few options for running the 4_ApplyTableExporter.py (Section 4.5) and 5_TrainingTableExporter.py (Section 4.6) scripts.

4.1.6 Modeling Options

This is where you define the predictor variables that you want to include in the Random Forest model, along with the parameters for the actual Random Forest run. Aside from the number of trees, we use the default parameters for the California SAGE implementation. Note that a large number of trees (usually more than ~100 trees) will often cause the model to fail.

4.1.7 Functions

These functions are used in several of the scripts.

4.2 1_GetLandsatWrapper.py

Options for this script are set in the Predictor Layer Options section of SAGE_Initialize.py.

The main purpose of this script is to create the annual Landsat composites, which contain spectral values from the Landsat imagery that are representative of peak of greenness. This is accomplished using one function call: getImagesLib.getLandsatWrapper(). This is a very versatile compositing function which allows the user to create their own, tailored composites if desired. If you are interested in compositing methods, I recommend that you spend some time with the getImagesLib package in geeViz.

1_GetLandsatWrapper.py has set as default all the options that were used in the California SAGE study. The composites use all available summer (June 1-September 30, or Julian day 152-273) Landsat 4, 5, 7, and 8 surface reflectance images, starting in 1984. The summer months are targeted because phreatophytes will be at their peak of greenness and will be more likely to be using groundwater resources during this time. The user may want to adjust the Julian days to reflect the greenest time of year for their specific geography. Clouds, cloud shadows, and snow are masked using the CFmask algorithm (Zhu and Woodcock, 2012, 2014; Zhu et al. 2015). The cloud score (Chastain et al. 2019) algorithm is another cloud masking option, and TDOM (Chastain et al. 2019) is a cloud shadow masking option that are available using the getLandsatWrapper() function.

Unless the user wants to create their own custom composites, they should only need to modify options in SAGE_Initialize.py. There are additional options in 1_GetLandsatWrapper.py should the user desire to make more specific decisions about how to create their composites. If geeViz is installed properly on the user's machine, the geeView() function should also bring up a tab on your browser where you can examine the annual composites as if using the GEE playground (this function is buggy, so don't panic if it doesn't work).

4.3 2_GetClimateWrapper.py

Options for this script are set in the Predictor Layer Options section of SAGE_Initialize.py.

The 2_GetClimateWrapper.py script collects annual mean climate data from NASA's Daily Surface Weather Data (Daymet) v4 collection (Thornton et al., 2016). The California SAGE study used daily minimum temperature, maximum temperature, and precipitation averaged over the water year (October 1-September 30, or Julian day 274-273), but the user is given the option to change these options. If choosing to average over the water year, make sure to specify a start year that is one year before the summer you are beginning the time series of Landsat composites (i.e., 1983 if Landsat composites begin in 1984). 2_GetClimateWrapper.py can be run at the same time as 1_GetLandsatWrapper.py since it does not depend on prior exports to run.

4.4 3_LandtrendrWrapper.py

Options for this script are set in the Predictor Layer Options section of SAGE_Initialize.py.

This script runs the LandTrendr temporal segmentation algorithm (Kennedy et al. 2010, 2018) on the Landsat composites and climate data composites, and it must be run after both 1_GetLandsatWrapper.py and 2_GetClimateWrapper.py have finished exporting. The code first brings in the Landsat and climate data composites and formats them correctly. It then runs the GEE ee.Algorithms.TemporalSegmentation.LandTrendr() function, and again formats the output for exporting. The preparatory and post-algorithm formatting are run using functions included in geeViz's changeDetectionLib.

4.5 4_ApplyTableExporter.py

Options for this script are set in the Training and Apply Data Options section of SAGE_Initialize.py.

4_ApplyTableExporter.py finds the value of each predictor layer created using 3LandtrendrWrapper.py for each year and band for all the GDEs. This is both the final input data that will be used to train the model (training GDEs) as well as the data that will be used to run the trained model on the entire set of GDEs (apply GDEs). Since the training GDEs are a subset of the apply GDEs, we first create this table for all the apply GDEs here. The next script in the series will use these tables to find the predictor values for the training GDEs.

To prepare, we first filter the apply GDEs using the minimum size threshold and do some formatting of the polygons, defined in the first section of the Training and Apply Data Options. Then, we add all the static predictor variables (ecoregion, HUC, etc.) using the addStrata() function.

Next, the getLT() function formats the LandTrendr outputs in the way that we want for our predictor layers, using the ltBands variable in the Training and Apply Data Options. There are five options here: the fitted values of the LandTrendr segments (.*_fitted), the magnitude of change of the entire segment (.*_mag), the year-to-year difference in value within the segment (.*_diff), the duration of the segment in years (.*_dur), and the slope of the segment (.*_slope). You can choose all or just a segment of these options, just make sure that you include all that you might potentially want to include as predictor layers (defined in the predictors variable in the Modeling Options of SAGE_Initialize.py). See Section 0 for a full explanation of how LandTrendr outputs work in this context.

Next, we export the tables to GEE asset. If you have multiple credentials at your disposal, you can set applyTablesUseMultiCredentials = True to split up the tables between credentials. If not, you can disregard this option.

Note that if you have a very large collection of GDEs (anecdotally, a collection of about 90,000 features seems to be about the maximum), Google Earth Engine may run out of memory during this step. If this occurs you may need to create these tables in batches. You can then either join them back together when running the model, or you can apply the trained model over the separate batches. Just make sure that all your training data is in one collection when training the model.

4.6 5_TrainingTableExporter.py

Options for this script are set in the Training and Apply Data Options section of SAGE_Initialize.py.

Here we export the training table, which must be done after the tables in 4_ApplyTableExporter.py have finished exporting. This is very similar to the apply tables created using the previous script in the series, however we are specifically focusing on the training GDEs – the subset of the apply GDEs that have DTG data from surrounding wells. Since the training GDEs are a subset of the apply GDE collection, we simply take the pre-exported table values from the apply tables here.

First, we filter and format the training GDEs, using the thresholds defined in the first section of the Training and Apply Data Options. Next, we loop through all the years and grab all the observations from the training GDEs that meet our criteria and join that GDE/year combination to the predictor layer observations collected in the apply tables.

4.7 6_ModelFitApply.py

Options for this script are set in the Modeling Options section of SAGE_Initialize.py.

Once all the prep work has been done using scripts 1-5, we use 6_ModelFitApply.py to run and apply the Random Forest model and export the predictions. The primary option you will need to consider is the ‘predictors’ variable – which predictors (LandTrendr outputs and static strata like ecoregion or HUC) you want to use to train and apply your model.

The script first brings in the training table, created using the previous script in the series, then trains the Random Forest model on it using the fitRFModel() function. Next, it exports information about the model (variable importance, out-of-bag error, etc.) to a json file and makes a simple graph showing variable importance. These are saved on your local computer in the path specified with the outputLocalRFModelInfoDir variable in SAGE_Initialize.py.

Finally, the script uses your trained model to predict DTG for each of the apply years on all of the apply GDEs. It uses the apply tables, created in 4_ApplyTableExporter.py, as the information with which to apply the model.

4.8 7_DownloadOutputs.py

Options for this script are set in the Modeling Options section of SAGE_Initialize.py.

This is a simple script that loops through all of your exported Prediction Tables and exports them to Google Drive.

4.9 8_TrendSummaries.py

Options for this script are set in the Modeling Options section of SAGE_Initialize.py.

After you have run the model and downloaded the outputs to your local computer, you can use this script as a framework to run summaries of the depth to groundwater trends. Note that it will probably not work out of the box, as it is designed to summarize over the Hydroregions and Groundwater Basins used to model SAGE, but you can use it as a template.

5 Works Cited

California Department of Fish and Wildlife (CDFW) (2021). Natural Communities. Available at: https://wildlife.ca.gov/Data/VegCAMP/Natural- Communities (Accessed September 27, 2021).

Chastain, R., Housman, I., Goldstein, J., Finco, M., and Tenneson, K. (2019). Empirical cross sensor comparison of Sentinel-2A and 2B MSI, Landsat-8 OLI, and Landsat-7 ETM top of atmosphere spectral characteristics over the conterminous United States. In Remote Sensing of Environment (Vol. 221, pp. 274-285). https://doi.org/10.1016/j.rse.2018.11.012.

Kennedy, R. E., Yang, Z., and Cohen, W. B. (2010). Detecting Trends in forest Disturbance and Recovery Using Yearly Landsat Time Series: 1. LandTrendr - Temporal Segmentation Algorithms. Remote Sensing Environ. 114, 2897–2910. doi:10.1016/j.rse.2010.07.008

Kennedy, R., Yang, Z., Gorelick, N., Braaten, J., Cavalcante, L., Cohen, W., et al. (2018). Implementation of the LandTrendr Algorithm on Google Earth Engine. Remote Sensing 10, 691. doi:10.3390/rs10050691

Klausmeyer, K., Howard, J., Keeler-Wolf, T., Davis-Fadtke, K., Hull, R., and Lyons, A. (2018). Mapping Indicators of Groundwater Dependent Ecosystems in California: Methods Report. San Francisco: California. Available at: https:// groundwaterresourcehub.org/public/uploads/pdfs/iGDE_data_paper_ 20180423.pdf.

Rohde M. M., T. Biswas, I. W. Housman, L. S. Campbell, K. R. Klausmeyer, and J. K. Howard, 2021: A Machine Learning Approach to Predict Groundwater Levels in California Reveals Ecosystems at Risk. Frontiers in Earth Sciences, 9. https://doi.org/10.3389/feart.2021.784499.

Thornton, P. E., Thornton, M. M., Mayer, B. W., Wei, Y., Devarakonda, R., Vose, R. S., et al. (2016). Daymet: Daily Surface Weather Data on a 1-km Grid for North America, Version 3. Oak Ridge, Tennessee, USA: ORNL DAAC. doi:10.3334/ ORNLDAAC/1328

Zhu, Z., & Woodcock, C. E. (2012). Object-based cloud and cloud shadow detection in Landsat imagery. In Remote Sensing of Environment (Vol. 118, pp. 83–94). https://doi.org/10.1016/j.rse.2011.10.028

Zhu, Z., & Woodcock, C. E. (2014). Continuous change detection and classification of land cover using all available Landsat data. In Remote Sensing of Environment (Vol. 144, pp. 152–171). https://doi.org/10.1016/j.rse.2014.01.011

Zhu, Z., Wang, S., and Woodcock, C. E. (2015). Improvement and Expansion of the Fmask Algorithm: Cloud, Cloud Shadow, and Snow Detection for Landsats 4-7, 8, and Sentinel 2 Images. Remote Sensing Environ. 159, 269–277. doi:10.1016/ j.rse.2014.12.014