

Using SoilGrids and Google Earth Engine with R to identify homogeneous zones

D G Rossiter

d.g.rossiter@cornell.edu

03-August-2021

Table of Contents

Introduction	2
Setup.....	2
Area of interest	4
Import SoilGrids layers	4
Stack and standardize.....	5
Visualize SoilGrids maps.....	8
PCA	10
Compute PCA	10
Eigenvalues and eigenvectors	11
Make PC images	15
Display properties of the PCs	15
Visualize PCs	16
Single PCs	16
R/G/B PC1/2/3.....	20
Clustering	21
Training points.....	21
Clustering – all PCs	23
Clustering – first 3 PCs.....	24
Exporting.....	26
For further processing in GEE	26
For further processing in R.....	27

Introduction

The purpose of this script is to identify relatively homogeneous areas with similar soil properties. These can be used as a pre-map for field work, or for stratified sampling. The homogeneous zones are identified by standardized Principal Components Analysis (PCA) of the stack of SoilGrids soil properties. These grids are accessed via Google Earth Engine (GEE), and the analysis is also performed there.

SoilGrids is a system for global digital soil mapping; see <https://www.isric.Org/explore/soilgrids> for more explanation. The key point is that SoilGrids predicts a set of soil properties over a set of standard depths everywhere on the non-ice, non-ocean, non-urban Earth.

This script computes, displays and (optionally) exports Principal Components of the SoilGrids v2.0 layers for multiple properties. There are six depth slices, according to `GlobalSoilMap.net` specifications. It also displays the PCA transformation, and then clusters the resulting PCs to identify relatively homogeneous zones.

This R Markdown document uses the `rgee` interface between R and GEE. It assumes you have set up this interface; see R Markdown document `ex_rgee.Rmd` for this. A major advantage of using `rgee` is that GEE is only used for the intensive imagery calculations, and normal small tasks (e.g., computing cumulative sums, making band names, making tables of summaries) can be done as usual in R.

This script saves the results of PCA in two places:

1. The input band names, means, standard deviations, PCA eigenvalues, and PCA eigenvectors (rotations) are saved in the connected directory of this script as `.rds` “R datafile”. This export is immediate, as the respective commands are executed.
2. The most important few PC images, the clustered image, and the clustering training points are exported to the user’s Google Drive, under the `rgee` folder. This export is staged as a GEE “task” and proceeds asynchronously. To use these in another script, check the task has completed by calling the `file.exists()` function on the file name.

Note that GEE knows the user’s identity after successful login with the `ee_initialize()` function; this is checked with the `ee_user_info()` function. These are included in the first code block, below.

Setup

Initialize the interface to GEE (via Python 3):

```
suppressMessages(library(reticulate))
Sys.which("python3") # is a V3 installed?

##           python3
## "/usr/local/bin/python3"
```

```
use_python(Sys.which("python3")) # use it
suppressMessages(library(rgee))
suppressMessages(library(rgeeExtra)) # https://github.com/r-
earthengine/rgeeExtra/blob/master/README.md
ee_check() # Check non-R dependencies

## ◎ Python version
## ✓ [Ok] /usr/local/bin/python3 v3.7
## ◎ Python packages:
## ✓ [Ok] numpy
## ✓ [Ok] earthengine-api

# ee_clean_credentials() # Remove credentials if you want to change the user
ee_clean_pyenv() # Remove reticulate system variables
ee_initialize()

## — rgee 1.0.9 ————— earthengine-api
0.1.262 —
## ✓ email: not_defined
## ✓ Initializing Google Earth Engine: ✓ Initializing Google Earth Engine:
DONE!
## ✓ Earth Engine user: users/cyrus621
##

—————

ee_user_info()

## ————— Earth Engine user
info —
## Reticulate python version:
## - /usr/local/bin/python3
## Earth Engine Asset Home:
## - users/cyrus621
## Credentials Directory Path:
## - /Users/rossiter/.config/earthengine//ndef/
## Google Drive Credentials:
## - NOT FOUND
## Google Cloud Storage Credentials:
## - NOT FOUND
##

—————

## $asset_home
## [1] "users/cyrus621"
##
## $user_session
## [1] "/Users/rossiter/.config/earthengine//ndef/"
##
## $gd_id
```

```
## [1] "NOT FOUND"
##
## $gcs_file
## [1] "NOT FOUND"
```

Other packages used in this script:

```
suppressMessages(library(tidyverse))
suppressMessages(library(terra))      # gridded data structures
```

Area of interest

Adjust the following to your area of interest.

1. The Region of Interest (roi) is specified as (long_w, lat_s, long_e, lat_n), i.e., as (lower-left, upper-right), in WGS84 geographic coordinates.
2. The Point of Interest (poi) is the centre of the map display, as (long, lat), in WGS84 geographic coordinates.

```
# a region of interest for map display and export, a rectangle in WGS84
Long/Lat
roi <- ee$Geometry$BBox(-77.6, 41.5, -75.5, 43)
# a centre point of interest for map display
poi <- ee$Geometry$Point(-76.5, 42.3)
```

Import SoilGrids layers

The list of property names and units is at

https://git.wur.nl/isric/soilgrids/soilgrids.Notebooks/-/blob/master/markdown/access_on_gee.md

Here we get pointers to the image stacks representing each property, clipped to the ROI. Note that these are on the GEE server and are never imported to the local R environment.

We omit silt, since we already have clay and sand. We omit total N and use only SOC concentration not SOC density nor SOC stocks.

```
SGc <- ee$Image("projects/soilgrids-isric/clay_mean")$clip(roi) # clay
SGs <- ee$Image("projects/soilgrids-isric/sand_mean")$clip(roi) # sand
SGbd <- ee$Image("projects/soilgrids-isric/bdod_mean")$clip(roi) # bulk
density
SGsoc <- ee$Image("projects/soilgrids-isric/soc_mean")$clip(roi) # SOC
SGpH <- ee$Image("projects/soilgrids-isric/phh2o_mean")$clip(roi) # pH H2O
SGcec <- ee$Image("projects/soilgrids-isric/cec_mean")$clip(roi) # CEC
SGcf <- ee$Image("projects/soilgrids-isric/cfvo_mean")$clip(roi) # coarse
fragments
```

Stack and standardize

Stack the bands into one multi-band image, also still on the GEE server:

```
SG <-  
SGc$addBands(SGs)$addBands(SGbd)$addBands(SGsoc)$addBands(SGpH)$addBands(SGcec)  
c)$addBands(SGcf)
```

Display information about this image stack. Note the use of `getInfo()` to force GEE to return the information so that it can be printed by R; otherwise the workspace object is just a pointer into GEE.

```
class(SG)  
  
## [1] "ee.image.Image"                "ee.element.Element"  
## [3] "ee.computedobject.ComputedObject" "ee.encodable.Encodable"  
## [5] "python.builtin.object"  
  
metadata <- SG$propertyNames()  
cat("Metadata: ", paste(metadata$getInfo(), "\n", collapse = " "))  
  
## Metadata: system:footprint  
## Transformation  
## Covariates  
## WoSIS_version  
## description  
## system:id  
## Mtry  
## Litter_layers  
## Mapped_units  
## title  
## Number_trees  
## Outputs_version  
## Model  
## system:version  
## Code_version  
## Model_type  
## system:asset_size  
## system:bands  
## system:band_names  
  
# one metadata item  
WoSIS_version <- SG$get('WoSIS_version')  
cat("WoSIS version used in this map:",  
    paste(WoSIS_version$getInfo(), "\n", collapse=""))  
  
## WoSIS version used in this map: Data stream 7  
  
inputBandNames <- SG$bandNames()$getInfo()  
cat("Input band names:\n")  
  
## Input band names:
```

```

print(inputBandNames)

## [1] "clay_0-5cm_mean"      "clay_5-15cm_mean"    "clay_15-30cm_mean"
## [4] "clay_30-60cm_mean"   "clay_60-100cm_mean" "clay_100-200cm_mean"
## [7] "sand_0-5cm_mean"     "sand_5-15cm_mean"   "sand_15-30cm_mean"
## [10] "sand_30-60cm_mean"  "sand_60-100cm_mean" "sand_100-200cm_mean"
## [13] "bdod_0-5cm_mean"    "bdod_5-15cm_mean"   "bdod_15-30cm_mean"
## [16] "bdod_30-60cm_mean"  "bdod_60-100cm_mean" "bdod_100-200cm_mean"
## [19] "soc_0-5cm_mean"     "soc_5-15cm_mean"    "soc_15-30cm_mean"
## [22] "soc_30-60cm_mean"   "soc_60-100cm_mean"  "soc_100-200cm_mean"
## [25] "phh2o_0-5cm_mean"   "phh2o_5-15cm_mean"  "phh2o_15-30cm_mean"
## [28] "phh2o_30-60cm_mean" "phh2o_60-100cm_mean" "phh2o_100-200cm_mean"
## [31] "cec_0-5cm_mean"     "cec_5-15cm_mean"    "cec_15-30cm_mean"
## [34] "cec_30-60cm_mean"   "cec_60-100cm_mean"  "cec_100-200cm_mean"
## [37] "cfvo_0-5cm_mean"    "cfvo_5-15cm_mean"   "cfvo_15-30cm_mean"
## [40] "cfvo_30-60cm_mean"  "cfvo_60-100cm_mean" "cfvo_100-200cm_mean"

dimOne <- length(inputBandNames)
cat("Number of input bands:", dimOne)

## Number of input bands: 42

proj <- SG$select("bdod_0-5cm_mean")$projection()
cat("Projection: ", paste(proj$getInfo(),"\n", collapse = " "))

## Projection: Projection
## PROJCS["World_Mollweide",
##   GEOGCS["GCS_WGS_1984",
##     DATUM["D_WGS_1984",
##       SPHEROID["WGS_1984", 6378137.0, 298.257223563]],
##     PRIMEM["Greenwich", 0.0],
##     UNIT["degree", 0.017453292519943295],
##     AXIS["Longitude", EAST],
##     AXIS["Latitude", NORTH]],
##   PROJECTION["Mollweide"],
##   PARAMETER["semi_minor", 6378137.0],
##   PARAMETER["false_easting", 0.0],
##   PARAMETER["false_northing", 0.0],
##   PARAMETER["central_meridian", 0.0],
##   UNIT["m", 1.0],
##   AXIS["x", EAST],
##   AXIS["y", NORTH]]
## list(250, 0, -17960089.4455105, 0, -250, 8697788.13163898)

scale <- SG$select("bdod_0-5cm_mean")$projection()$nominalScale()
cat("Nominal scale: ", scale$getInfo())

## Nominal scale: 250

dims <- SG[[1]]$getInfo()$bands[[1]]$dimensions
cat("Image dimensions:", unlist(dims))

```

```
## Image dimensions: 1078 663
```

Save the band names as an R object for use other R scripts:

```
saveRDS(inputBandNames, file = "./inputBandNames.rds")
```

To make the ranges of the different properties compatible, standardize each band to mean=0, s.d.=1.

First, compute the per-band means and s.d. with reduceRegion:

```
SGmean <- SG$reduceRegion(ee$Reducer$mean(),
  geometry = roi, scale = scale, bestEffort = TRUE)
head(SGmean$getInfo(),3) # an example of the means

## $`bdod_0-5cm_mean`
## [1] 117.7372
##
## $`bdod_100-200cm_mean`
## [1] 173.4357
##
## $`bdod_15-30cm_mean`
## [1] 137.2048

SGsd <- SG$reduceRegion(ee$Reducer$stdDev(),
  geometry = roi, scale = scale, bestEffort = TRUE)
head(SGsd$getInfo(),3) # an example of the s.d.'s

## $`bdod_0-5cm_mean`
## [1] 7.416615
##
## $`bdod_100-200cm_mean`
## [1] 3.974953
##
## $`bdod_15-30cm_mean`
## [1] 5.698883
```

Export these for use in further R scripts:

```
saveRDS(as.vector(SGmean$getInfo()), file = "./inputBandMeans.rds")
saveRDS(as.vector(SGsd$getInfo()), file = "./inputBandSDs.rds")
```

Convert the mean and s.d. to constant image stacks:

```
SGmean.img <- ee$Image$constant(SGmean$values(inputBandNames))
SGsd.img <- ee$Image$constant(SGsd$values(inputBandNames))
```

Now standardize; this applies the subtract and divide across the bands in order.

```
SGstd <- SG$subtract(SGmean.img)
SGstd <- SGstd$divide(SGsd.img)
```

Summarize the standardized variables and show the range of the first listed property at all depths, and the overall minimum and maximum:

```
SGstd.minMax <- SGstd$reduceRegion(ee$Reducer$minMax(),
  geometry = roi, scale = scale,
  maxPixels = 1e9, bestEffort = TRUE)
minMaxNames <- names(SGstd.minMax$getInfo())
minMaxVals <- SGstd.minMax$values()$getInfo()
# per property min/max
head(data.frame(property_depth_q=minMaxNames, value=minMaxVals), 12)

##           property_depth_q      value
## 1      bdod_0-5cm_mean_max  2.732088
## 2      bdod_0-5cm_mean_min -3.200538
## 3  bdod_100-200cm_mean_max  3.160880
## 4  bdod_100-200cm_mean_min -7.153708
## 5      bdod_15-30cm_mean_max  4.877299
## 6      bdod_15-30cm_mean_min -3.720877
## 7      bdod_30-60cm_mean_max  4.270982
## 8      bdod_30-60cm_mean_min -6.834904
## 9        bdod_5-15cm_mean_max  4.190182
## 10       bdod_5-15cm_mean_min -3.534867
## 11  bdod_60-100cm_mean_max  3.030865
## 12  bdod_60-100cm_mean_min -7.225288

# overall min/max
print(c(min(minMaxVals), max(minMaxVals)) )

## [1] -7.225288 62.233742

# check the means
# SGstd.mean <- SGstd$reduceRegion(ee$Reducer$mean(),
#   geometry = roi, scale = scale, maxPixels = 1e9, bestEffort = TRUE)
# meanVals <- SGstd.mean$values()$getInfo()
# per property means
# data.frame(property_depth_q=minMaxNames, value=meanVals)
```

In the example area some of the input bands are right-skewed, especially SOC and, to a lesser extent, CEC. This could be minimized by log-transforming or square-root transforming the property values. Another approach would be to normalize to $[-1 \dots + 1]$ by linear stretch.

Visualize SoilGrids maps

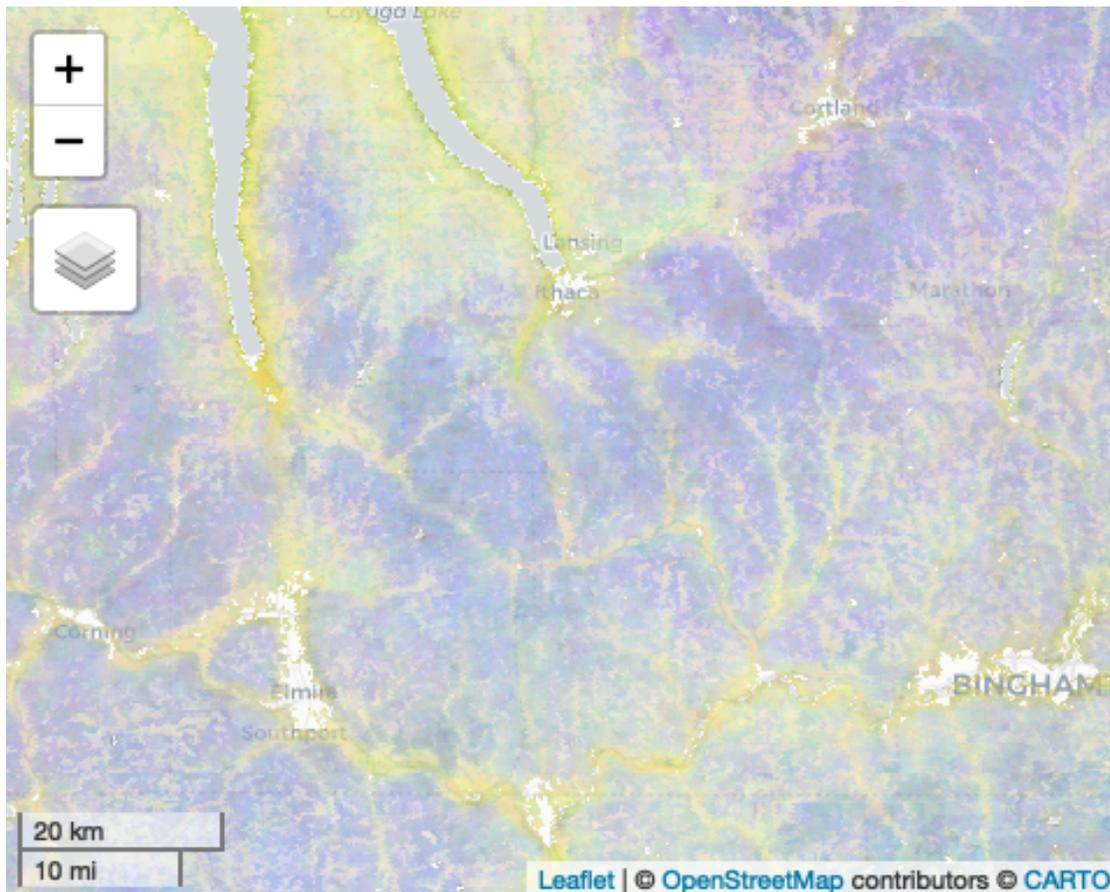
Any three bands of the 42-band stack can be displayed as a colour composite with the `Map$addLayer` function. Within `rgee` this uses [Leaflet](#) called from the Python interface to GEE.

Here, show a 3-colour map of three layers' standardized bulk densities:

```

range <- c(min(minMaxVals[c(2,4,6)]),
           min(minMaxVals[c(1,3,5)]))
imgViz <- list( # see ee.data.getMapId for valid parameters
  min = range[1],
  max = range[2],
  # R. G. B
  bands = c("bdod_0-5cm_mean", "bdod_15-30cm_mean", "bdod_60-100cm_mean"),
  gamma = c(0.95, 1.1, 1),
  opacity = 0.7
)
Map$centerObject(poi, zoom=9)
Map$addLayer(SGstd, imgViz, 'Bulk density 3 layers')

```



In this example the darker colours have the highest bulk density in the subsoil, which is represented by blue. These are mostly on the Appalachian Plateau and have either fragipans or dense till subsoils.

Show a 3-colour map of three standardized properties of one layer, here bulk density, clay and pH:

```

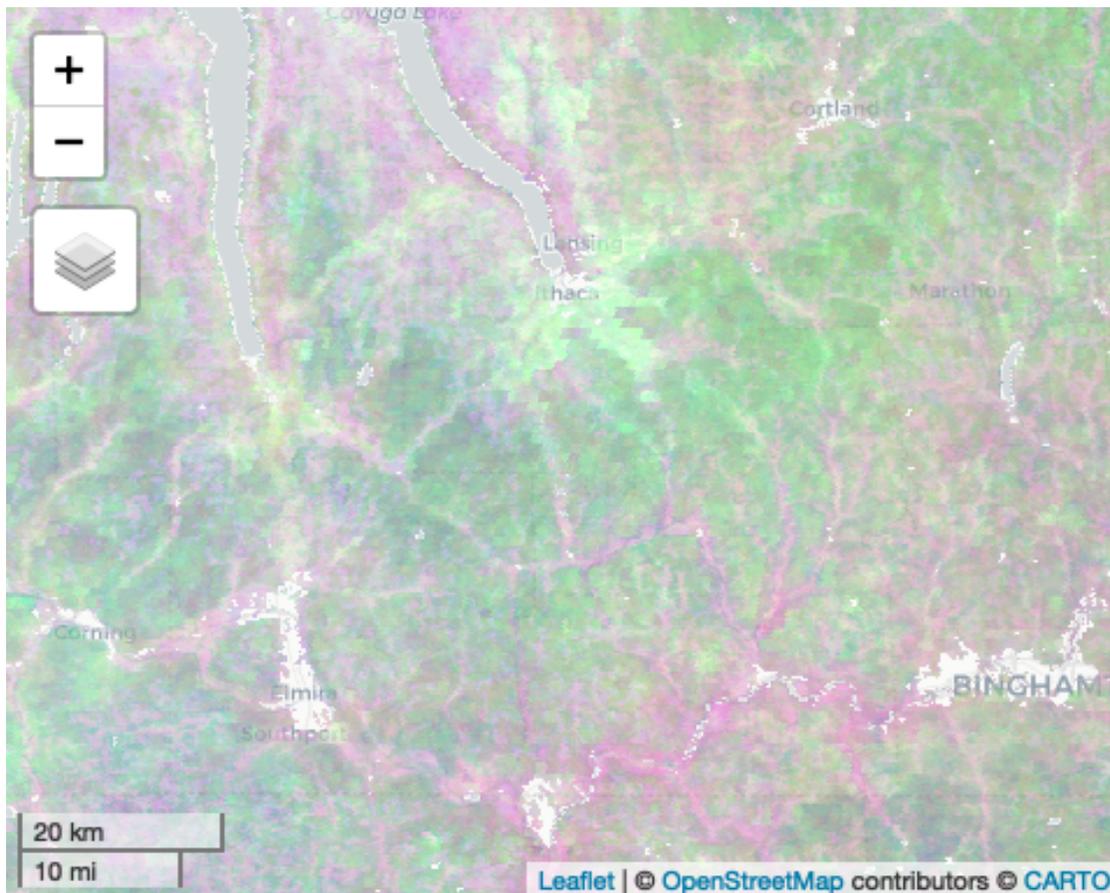
range <- c(min(minMaxVals[c(2,38,50)]),
           min(minMaxVals[c(1,37,49)]))
imgViz <- list(
  min = range[1],

```

```

max = range[2],
# R, G, B
bands = c("bdod_0-5cm_mean", "clay_0-5cm_mean", "phh2o_0-5cm_mean"),
gamma = c(0.95, 1.1, 1),
opacity = 0.7
)
Map$centerObject(poi, zoom=9)
Map$addLayer(SGstd, imgViz, 'Bulk density, clay, pH of top layer')

```



We can clearly see regionalization based on a set of properties. This is what we want to discover, but using all properties and depth slices.

Now we want to use all the properties to regionalize, via PCA.

PCA

Compute PCA

Compute the PCA for the SoilGrids property stack. The bands are already centred and scaled.

```

# convert image to an array of pixels, for matrix calculations
# dimensions are N x P; i.e., pixels x bands
arrays <- SGstd$array()
### print(arrays$getInfo())

# Compute the covariance of the bands within the region.
covar <- arrays$reduceRegion(
  ee$Reducer$centeredCovariance(),
  geometry = roi, scale = scale,
  maxPixels = 1e6,
  bestEffort = TRUE
)
### str(covar$getInfo())
### print(covar$getInfo())

# Get the covariance result and cast to an array.
# This represents the band-to-band covariance within the region.
covarArray <- ee$Array(covar$get('array'))

# note we know the dimensions from the inputs

# Perform an eigen analysis and slice apart the values and vectors.
eigens <- covarArray$eigen()
# the first item of each slice (PC) is the corresponding eigenvalue
# the remaining items are the eigenvalues (rotations) for that PC

```

Eigenvalues and eigenvectors

Extract the P-length vector of eigenvalues and show their proportion of the total variance:

```

# by removing the first axis (eigenvalues) and converting to a vector
# array$slice(axis=0, start=0, end=null, step=1)
# here we only have one axis, indexed by the PC
eigenValues <- eigens$slice(1L, 0L, 1L)
cat('Eigenvalues:')

## Eigenvalues:

print(eigenValues.vect <- unlist(eigenValues$getInfo()))

## [1] 15.469362158  7.918745262  4.745552270  2.891400472  2.160102069
## [6]  1.629330389  1.377544478  1.261781870  0.759056479  0.531884789
## [11]  0.440576395  0.387050855  0.330839293  0.264921276  0.240311958
## [16]  0.208602218  0.195436634  0.176815574  0.135210953  0.110162149
## [21]  0.098400016  0.093907485  0.085858010  0.075305332  0.049566784
## [26]  0.045170073  0.042921822  0.040665777  0.036937482  0.031677218
## [31]  0.025442880  0.024074020  0.020036003  0.016467759  0.015218777
## [36]  0.011552377  0.009770266  0.008553613  0.008098974  0.005493895
## [41]  0.004811953  0.002540593

```

```

# compute and show proportional eigenvalues
eigenValuesProportion <-
round(100*(eigenValues.vect/sum(eigenValues.vect)),2)
cat('PCs percent of variance explained:')

## PCs percent of variance explained:

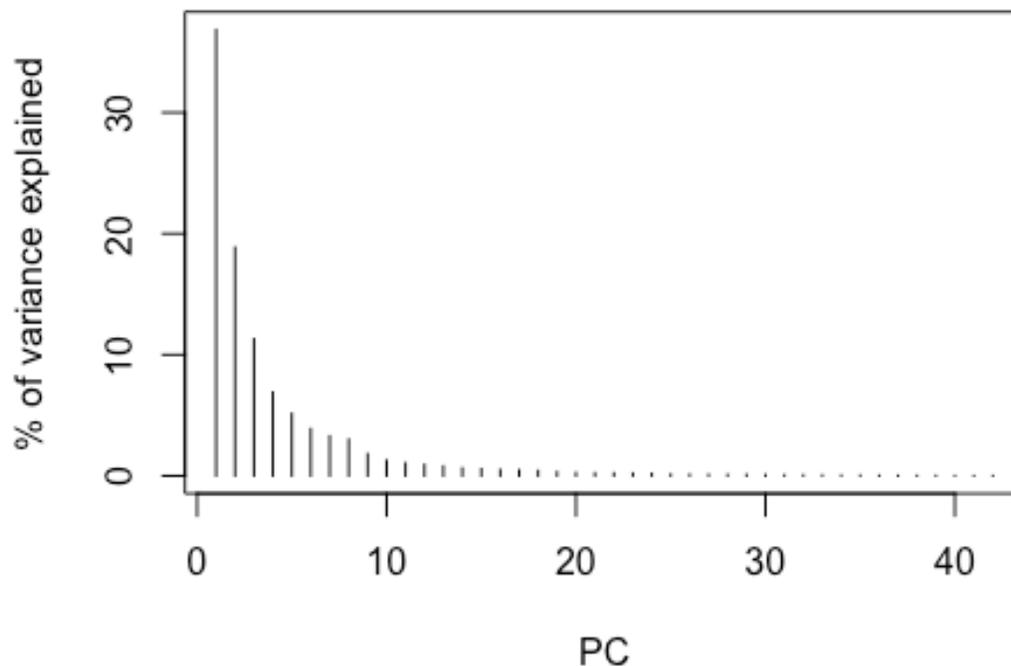
print(eigenValuesProportion)

## [1] 36.84 18.86 11.30 6.89 5.14 3.88 3.28 3.01 1.81 1.27 1.05
0.92
## [13] 0.79 0.63 0.57 0.50 0.47 0.42 0.32 0.26 0.23 0.22 0.20
0.18
## [25] 0.12 0.11 0.10 0.10 0.09 0.08 0.06 0.06 0.05 0.04 0.04
0.03
## [37] 0.02 0.02 0.02 0.01 0.01 0.01

plot(eigenValuesProportion, type="h", xlab="PC",
      ylab="% of variance explained",
      main="Standardized PCA")

```

Standardized PCA



```

evsum <- cumsum(eigenValuesProportion)
cat('PCs percent of variance explained, cumulative sum:')

## PCs percent of variance explained, cumulative sum:

```

```

print(round(evsum,1))

## [1] 36.8 55.7 67.0 73.9 79.0 82.9 86.2 89.2 91.0 92.3 93.3
94.2
## [13] 95.0 95.7 96.2 96.7 97.2 97.6 98.0 98.2 98.4 98.7 98.9
99.0
## [25] 99.2 99.3 99.4 99.5 99.6 99.6 99.7 99.8 99.8 99.9 99.9
99.9
## [37] 99.9 100.0 100.0 100.0 100.0 100.0

cat(npc95 <- which(evsum > 95)[1],
    "PCs are needed to explain 95% of the variance")

## 13 PCs are needed to explain 95% of the variance

```

Export the eigenvalues for use in other R scripts:

```
saveRDS(eigenValues.vect, file = "./eigenValuesVector.rds")
```

Extract the eigenvectors (rotations) and show the first few. Export them as an R datafile for use in other R scripts.

```
# The eigenvectors (rotations); this is a PxP matrix with eigenvectors in rows.
```

```
eigenVectors <- eigens$slice(1L, 1L)
```

```
# show the first few rotations
```

```
cat('Eigenvectors 1--3:')
```

```
## Eigenvectors 1--3:
```

```

print(data.frame(band=inputBandNames,
                 rotation1 = eigenVectors$getInfo()[[1]],
                 rotation2 = eigenVectors$getInfo()[[2]],
                 rotation3 = eigenVectors$getInfo()[[3]]
                ))

##           band      rotation1      rotation2      rotation3
## 1   clay_0-5cm_mean -0.178317977 -0.14079428  0.0562706335
## 2   clay_5-15cm_mean -0.174667912 -0.18903655  0.0119594233
## 3   clay_15-30cm_mean -0.181733188 -0.18669802  0.0148812581
## 4   clay_30-60cm_mean -0.182122683 -0.18705778  0.0005602095
## 5   clay_60-100cm_mean -0.172534342 -0.20224699 -0.0061147415
## 6   clay_100-200cm_mean -0.159745179 -0.20138042 -0.0019767387
## 7   sand_0-5cm_mean  0.155512294  0.17542640  0.0641614537
## 8   sand_5-15cm_mean  0.157863341  0.20282052  0.0695266478
## 9   sand_15-30cm_mean  0.164405750  0.20914975  0.0692816109
## 10  sand_30-60cm_mean  0.176076300  0.21193354  0.0718742444
## 11  sand_60-100cm_mean  0.175269062  0.22194233  0.0698899934
## 12  sand_100-200cm_mean  0.160396498  0.23429293  0.0804221702
## 13  bdod_0-5cm_mean  -0.173785089  0.16000018  0.0018684265
## 14  bdod_5-15cm_mean -0.164105925  0.18410879  0.0248733900

```

```

## 15    bdod_15-30cm_mean -0.149424770  0.19839200  0.0045661072
## 16    bdod_30-60cm_mean -0.138994692  0.06930870 -0.0759328450
## 17    bdod_60-100cm_mean  0.083323574 -0.16862402 -0.1227287340
## 18    bdod_100-200cm_mean  0.112034443 -0.16178122 -0.0833157243
## 19      soc_0-5cm_mean  0.209677711 -0.06516654  0.0596968029
## 20      soc_5-15cm_mean  0.141941009 -0.14643776  0.1801388443
## 21      soc_15-30cm_mean  0.095544425 -0.10515729  0.2425924431
## 22      soc_30-60cm_mean  0.084363960 -0.10320036  0.2421127774
## 23      soc_60-100cm_mean  0.080991362 -0.08665358  0.1985877085
## 24      soc_100-200cm_mean  0.088944062 -0.08798651  0.1931017671
## 25      phh2o_0-5cm_mean -0.221694064  0.07677400 -0.0161625188
## 26      phh2o_5-15cm_mean -0.222231643  0.08647110 -0.0082195672
## 27      phh2o_15-30cm_mean -0.222519269  0.08725812  0.0061001970
## 28      phh2o_30-60cm_mean -0.223377191  0.07561581  0.0363480376
## 29      phh2o_60-100cm_mean -0.225100991  0.05053551  0.0439076183
## 30      phh2o_100-200cm_mean -0.224543000  0.02369451  0.0262389679
## 31      cec_0-5cm_mean  0.180587409 -0.02581085  0.0867016182
## 32      cec_5-15cm_mean  0.071673714 -0.12392392  0.2928519997
## 33      cec_15-30cm_mean  0.058254395 -0.15242153  0.2872644040
## 34      cec_30-60cm_mean -0.014396312 -0.19431154  0.2194515881
## 35      cec_60-100cm_mean  0.005565774 -0.21334321  0.1814780640
## 36      cec_100-200cm_mean  0.030120478 -0.20107190  0.1924610452
## 37      cfvo_0-5cm_mean  0.011581924 -0.18797230 -0.1746220856
## 38      cfvo_5-15cm_mean  0.131213709 -0.15114204 -0.2707702445
## 39      cfvo_15-30cm_mean  0.131791408 -0.15793369 -0.2664900021
## 40      cfvo_30-60cm_mean  0.146740558 -0.12494069 -0.2742181137
## 41      cfvo_60-100cm_mean  0.146433505 -0.09393983 -0.2791610091
## 42      cfvo_100-200cm_mean  0.143194675 -0.08466486 -0.2852123717

```

```
# export the eigenvectors
```

```
eVm <- matrix(unlist(eigenVectors$getInfo()), byrow = TRUE, nrow = dimOne)
saveRDS(eVm, file = "./eigenvectorMatrix.rds")
```

The eigenvalues (rotations) can be interpreted as the contribution of each input band to the PC. The sign is arbitrary, we are looking for which bands contribute the most (largest absolute values) and which in the same sense and which in the opposite sense.

These must be interpreted for each area, and can not be extrapolated.

In the example area, PC1 is dominated by clay opposed to sand, bulk density in the same sense as clay, pH also in the same sense as clay, and coarse fragments in the opposite sense. As expected, all layers of a property contribute about the same; contrasts will be found in higher PCs. So PC1 groups clayey, higher pH, denser soils with few coarse fragments, and their opposite. SOC and CEC hardly contribute to PC1.

In the example area, PC2 represents the contrast between high SOC and CEC soils, and their opposite. Note this is orthogonal to PC, so these contrasts are mostly independent of those which contribute to PC1. Note also the contrast between bulk densities at different depths. Deeper layers contribute more to this PC than shallower layers, and this is correlated in this PC with low CEC and SOC.

Make PC images

To compute the PC images, first convert the array image to a 2D array image for matrix computations, along the single axis.

```
arrayImage <- arrays$toArray(1L)
```

Second, left multiply the image array (P X P) by the matrix of eigenvectors (P X N), to get a PxN matrix, now in the rotated vector space.

```
PCsMatrix <- ee$Image(eigenVectors)$matrixMultiply(arrayImage)
```

Third, convert the PCs array into a single P-band image, with appropriate band names.

```
# arrayProject: "Projects the array in each pixel to a lower dimensional #  
space by specifying the axes to retain"  
# arrayFlatten: "Converts a single band image of equal-shape  
#  
                multidimensional pixels  
#  
                to an image of scalar pixels,  
#  
                with one band for each element of the array."  
PCbandNames <- as.list(paste0('PC', seq(1L:dimOne)))  
PCs <- PCsMatrix$arrayProject(list(0L))$arrayFlatten(  
  list(PCbandNames)  
)
```

From here on, only work with the "significant" PCs.

```
PCs95 <- PCs$select(0L:(npc95-1L)) # indexing starts from 0  
PCs95$bandNames()$getInfo()
```

```
## [1] "PC1" "PC2" "PC3" "PC4" "PC5" "PC6" "PC7" "PC8" "PC9" "PC10"  
## [11] "PC11" "PC12" "PC13"
```

Display properties of the PCs

Means and standard deviations:

```
means <- PCs95$reduceRegion(ee$Reducer$mean(),  
  geometry = roi, scale = scale, bestEffort = TRUE,  
  maxPixels = 1e6)  
cat('Statistics of the PCs: mean:')  
  
## Statistics of the PCs: mean:  
  
means$values()$getInfo()  
  
## [1] -0.0301727456 -0.0003252514 0.0087302167 0.0006943932 -0.0081394281  
## [6] -0.0028221798 -0.0081468485 -0.0315038939 -0.0207919531 -0.0758273996  
## [11] -0.0270154465 -0.0032222143 -0.0166398025  
  
sds <- PCs95$reduceRegion(ee$Reducer$stdDev(),  
  geometry = roi, scale = scale, bestEffort = TRUE,
```

```

                                maxPixels = 1e6)
cat('Statistics of the PCs: s.d.:')
## Statistics of the PCs: s.d.:
sds$values()$getInfo()
## [1] 3.8665518 0.6860347 0.5835777 0.5742136 0.5186738 2.7571690 2.1082134
## [8] 1.6303097 1.3756857 1.2210845 1.1164366 1.0412499 0.8076193

```

The PCs could be standardized to have similar importance in a composite or a cluster analysis. But since the PCs are in order of variance explained, we want to retain this sequence. Note however that these standard deviations are not necessarily monotonically decreasing.

Visualize PCs

Single PCs

Display the first PC. First, find its dynamic range:

```

# get display range for the first PC
PC1 <- PCs$select('PC1') # can also write PCs[[1]]
rangeDict <- PC1$reduceRegion(
  reducer = ee$Reducer$minMax(),
  bestEffort = TRUE
)
print(range <- rangeDict$getInfo())
## $PC1_max
## [1] 22.29575
##
## $PC1_min
## [1] -14.14187

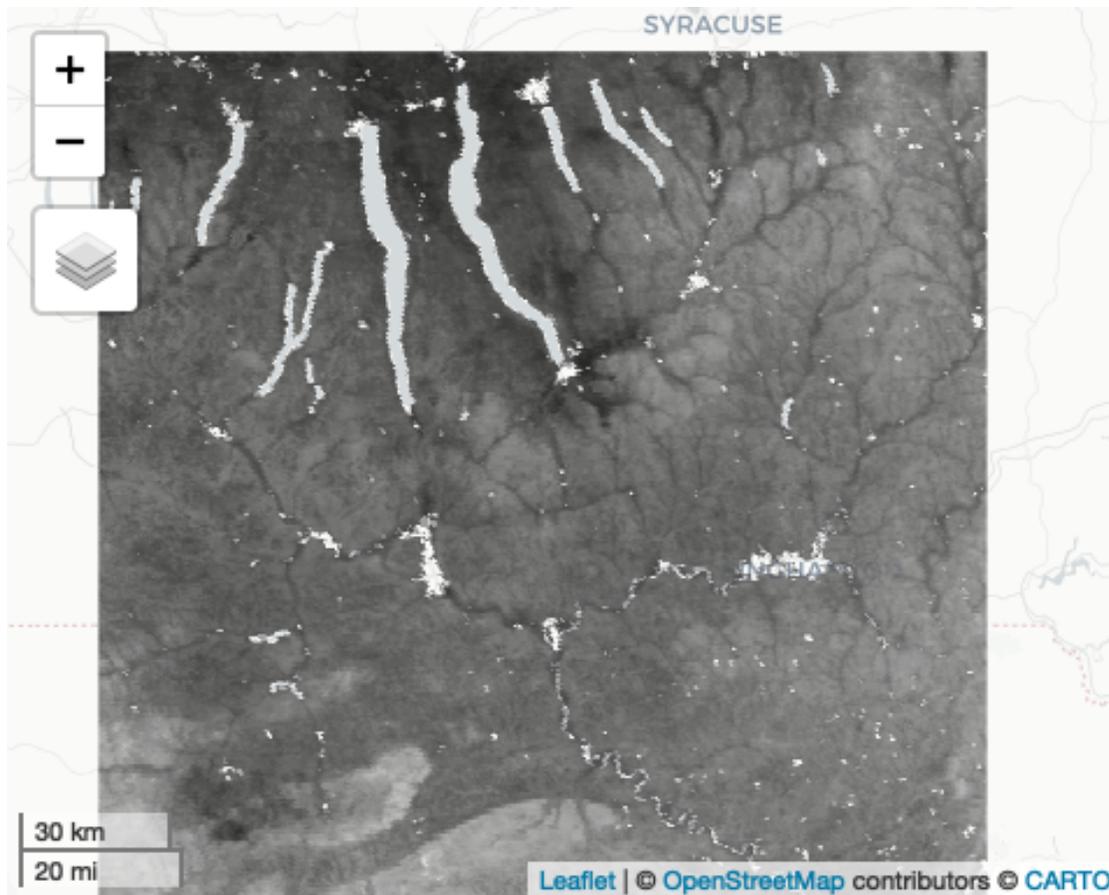
```

Use this range to set the visualization and display the map.

```

imgViz <- list(
  min = range[[2]],
  max = range[[1]],
  bands = c("PC1"),
  gamma = c(0.95),
  opacity = 0.9
)
Map$centerObject(poi, zoom=8)
Map$addLayer(PCs, imgViz)

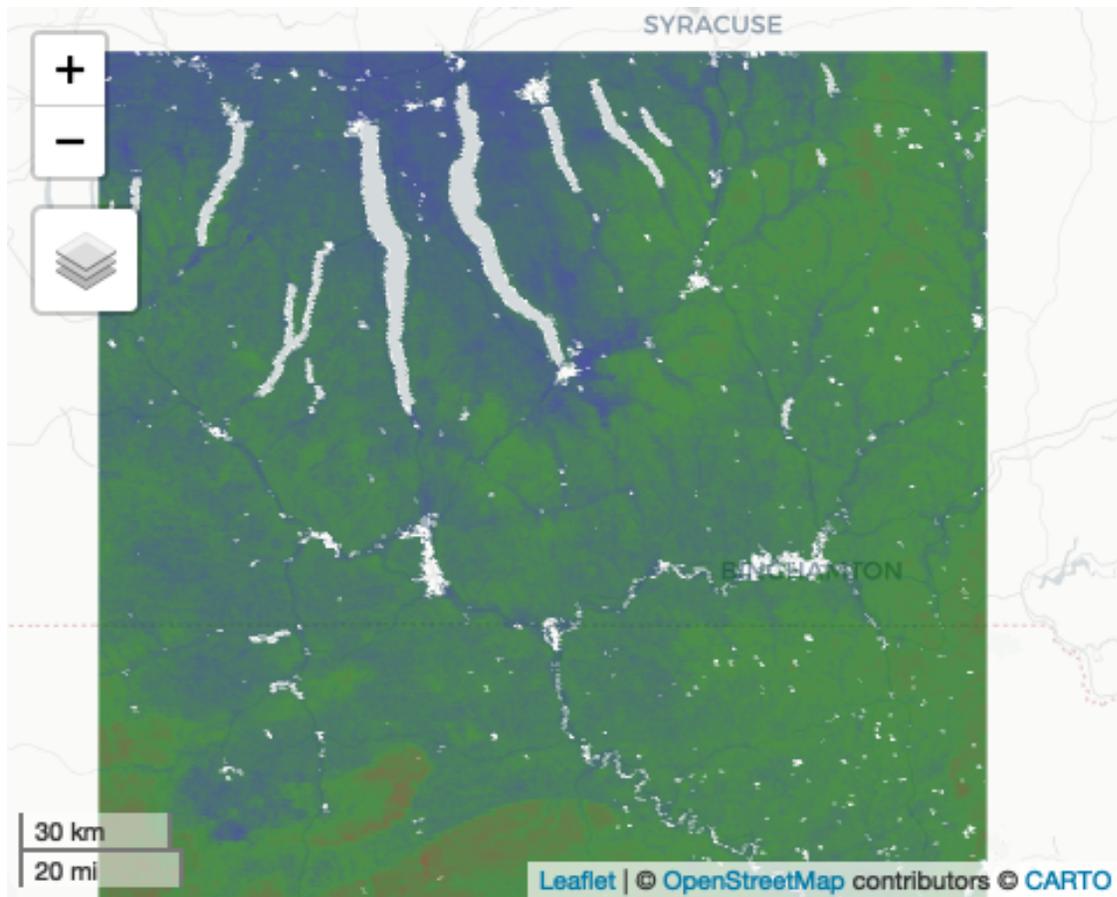
```



In the example area, this shows the contrast of PC1 discussed above, i.e., clayey, higher pH, denser soils with few coarse fragments (darker gray) in the Lake Ontario plain and in some valleys, and their opposite (lighter gray) covering most of the Appalachian Plateau.

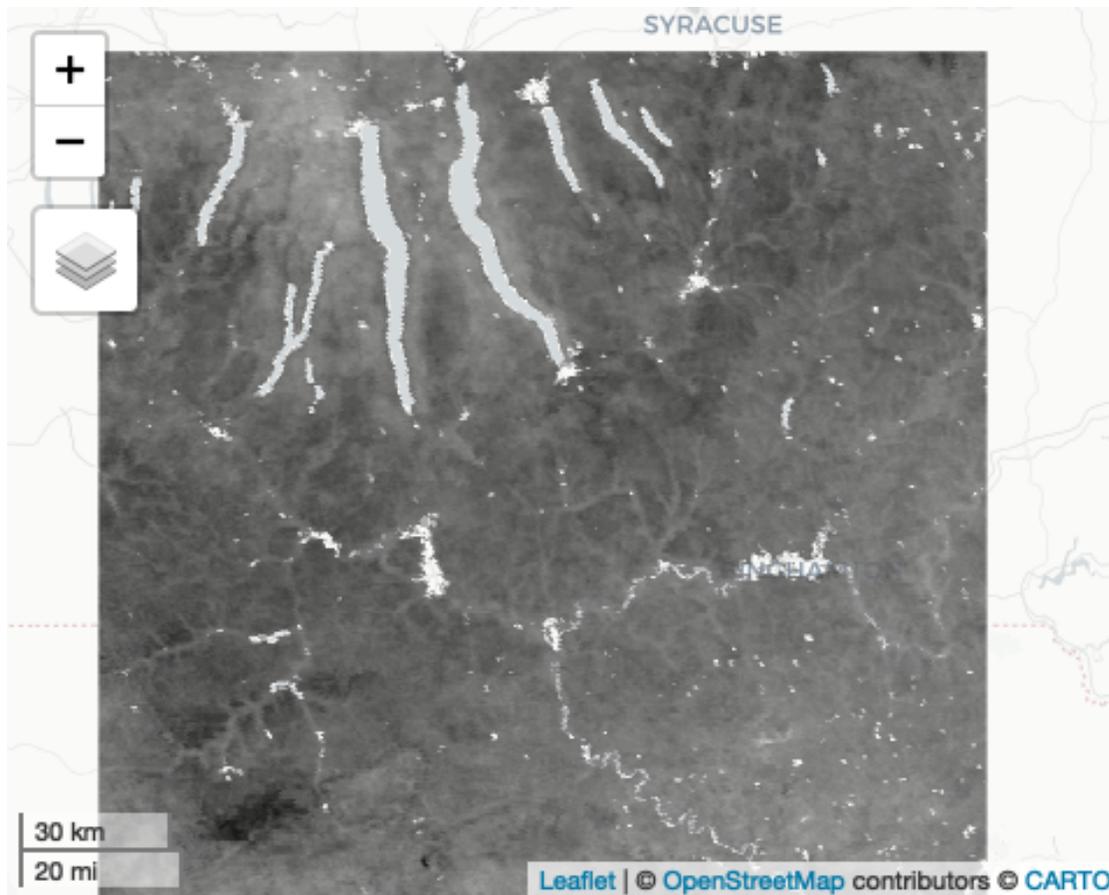
We can also display a single band of an image in color. To do this we define the palette parameter with a color ramp represented by a list of [hexidecimal color codes](#) which represent RGB triplets. Many of these also have names.

```
imgViz <- list(
  min = range[[2]],
  max = range[[1]],
  bands = c("PC1"),
  palette = c('darkblue', 'darkgreen', 'darkred'),
  opacity = 0.7
)
Map$centerObject(poi, zoom=8)
Map$addLayer(PCs, imgViz)
```



Similarly for PC2:

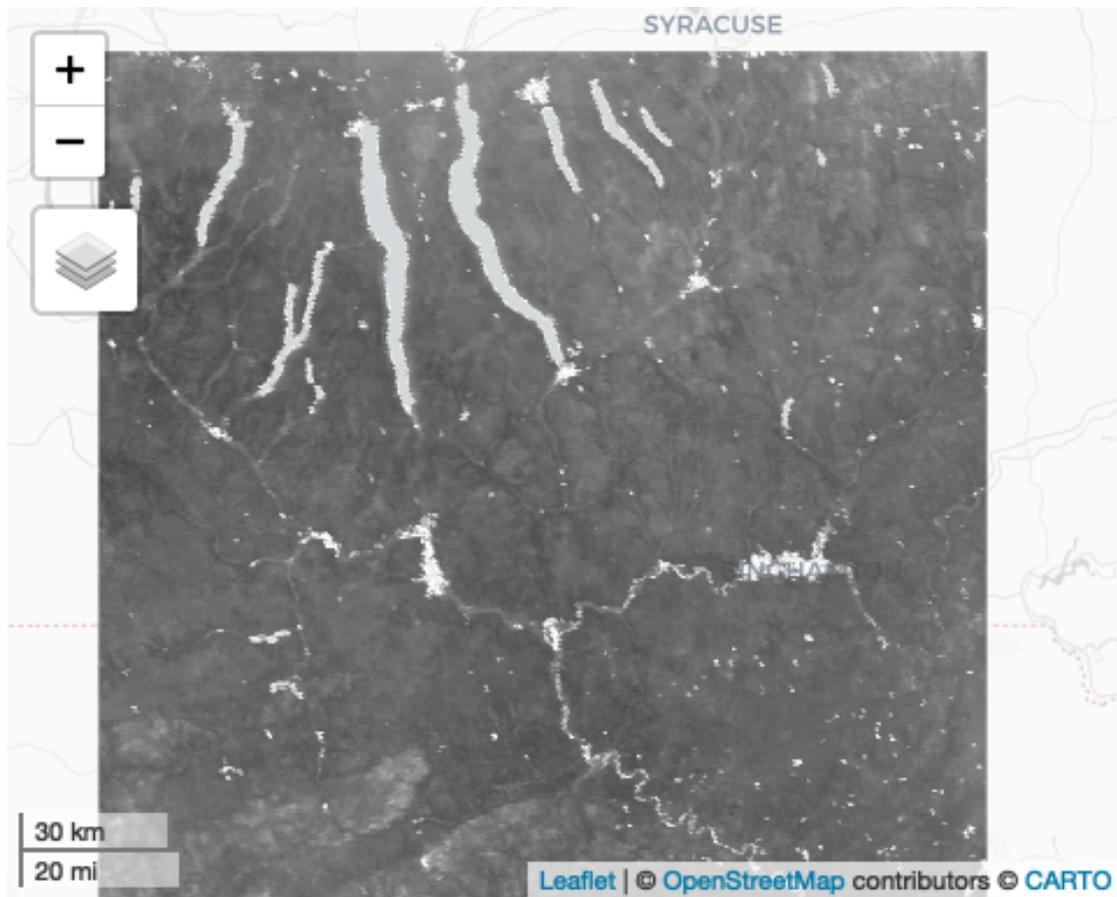
```
imgViz <- list(  
  min = range[[2]],  
  max = range[[1]],  
  bands = c("PC2"),  
  gamma = c(0.95),  
  opacity = 0.9  
)  
Map$centerObject(poi, zoom=8)  
Map$addLayer(PCs, imgViz)
```



Here the darker greys are high SOC and (partly because of this) CEC soils, mostly notably around Lake Oneida, in the Mount Morris valley, and on some of the higher hills to the south.

Similarly for PC3:

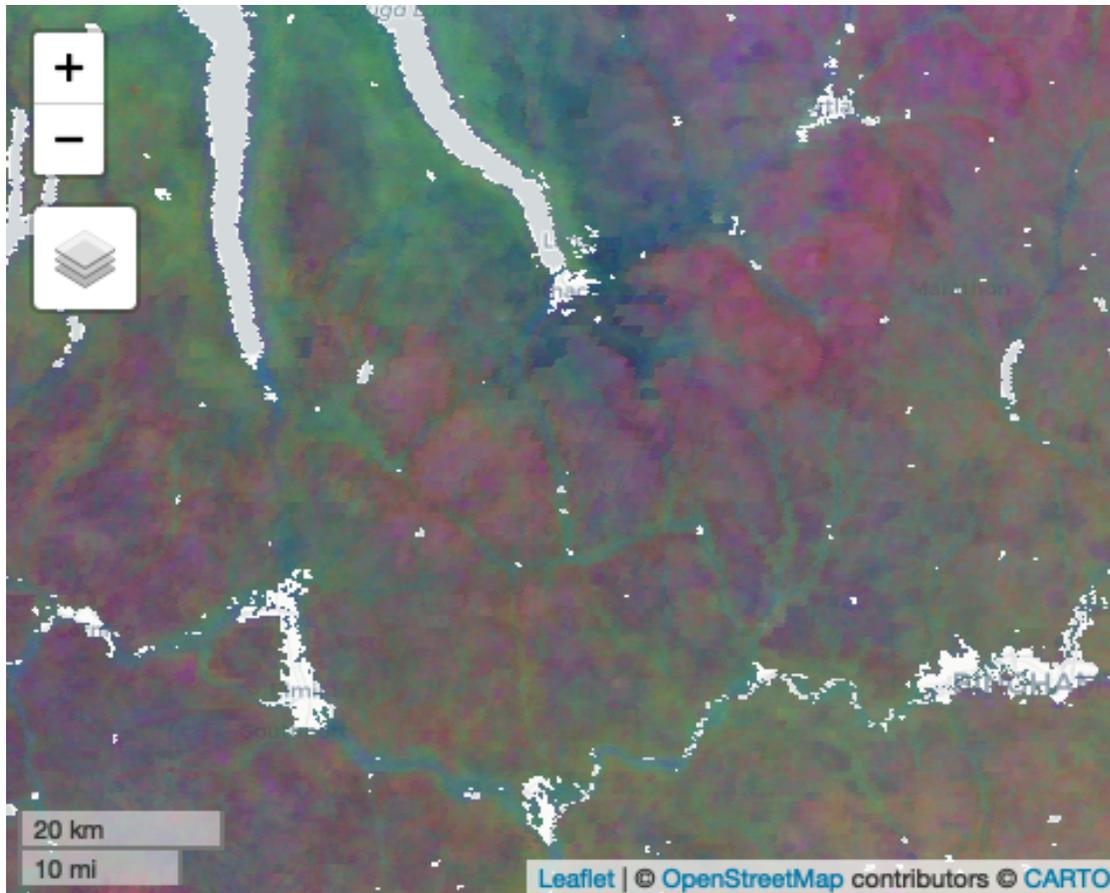
```
imgViz <- list(
  min = range[[2]],
  max = range[[1]],
  bands = c("PC3"),
  gamma = c(0.95),
  opacity = 0.9
)
Map$centerObject(poi, zoom=8)
Map$addLayer(PCs, imgViz)
```



R/G/B PC1/2/3

Display an R/G/B composite of the first 3 PCs, using the dynamic range from PC1:

```
imgViz <- list(
  min = range[[2]],
  max = range[[1]],
  bands = c("PC1", "PC2", "PC3"),
  gamma = c(0.95),
  opacity = 0.9
)
Map$centerObject(poi, zoom=9)
Map$addLayer(PCs, imgViz)
```



This very nicely shows a regionalization. In the next section we convert this visualization to objectively-defined clusters of similar soils.

Clustering

Now these PCs can be clustered. GEE has several procedures for this, taken from the [Weka](#) project. The usual K-means classifier is `ee.Clusterer.wekaKMeans`, which requires the analyst to set the number of clusters.

Training points

Set up a set of training pixels as a `FeatureCollection`.

```
training <- PCs95$sample(
  region = roi,
  scale = scale,
  tileSize = 16, # spread the computation over several processors
  numPixels = 10000L,
  geometries = TRUE # also record the coordinates
)
training$size()$getInfo()
```

```
## [1] 9667
```

```
# show locations. Only visualization option for points is colour  
# Map$addLayer(training, {list(color="lightblue")}, "training points")
```

Some of the requested 10k points were at “nodata” locations (water, urban) and so were not recorded.

Here is the first training point: its coordinates and the values of the PCs at that point.

```
training$first()$getInfo()
```

```
## $type  
## [1] "Feature"  
##  
## $geometry  
## $geometry$type  
## [1] "Point"  
##  
## $geometry$coordinates  
## [1] -76.10489 42.08390  
##  
##  
## $id  
## [1] "0"  
##  
## $properties  
## $properties$PC1  
## [1] 0.9679998  
##  
## $properties$PC10  
## [1] -0.7178974  
##  
## $properties$PC11  
## [1] -0.2593502  
##  
## $properties$PC12  
## [1] -0.2741765  
##  
## $properties$PC13  
## [1] 0.004559222  
##  
## $properties$PC2  
## [1] 1.898987  
##  
## $properties$PC3  
## [1] -1.395087  
##  
## $properties$PC4  
## [1] 1.796083  
##
```

```
## $properties$PC5
## [1] 0.2609544
##
## $properties$PC6
## [1] 1.614867
##
## $properties$PC7
## [1] 1.804215
##
## $properties$PC8
## [1] 0.8353943
##
## $properties$PC9
## [1] -0.1551228
```

Clustering – all PCs

Use this training sample to build clusters. Use the “Manhattan” distance between points in multivariate space (not the default “Euclidean”) to reduce the influence of unusual values of the PCs.

In k-means clustering the analyst must specify the desired number of clusters. This should correspond to the degree of detail: more clusters are more internally-homogeneous but some classes may be very close. This could be based on the expected number of general soil-forming environments in the region of interest. Another approach is to specify a large number of clusters and then group them by the analyst’s expert knowledge, or else reduce the number until no “artefacts” are seen.

```
n_clusters = 8
clusters <- ee$Clusterer$
  wekaKMeans(nClusters=n_clusters, distanceFunction="Manhattan")$
  train(training)
```

Now use these to classify the PC stack to an image with the cluster number, and show the proportion in each cluster. Here each pixel in the stack is assigned to its closest cluster by Manhattan distance.

```
PCcluster <- PCs95$cluster(clusters)
hist <- PCcluster$reduceRegion(
  reducer = ee$Reducer$frequencyHistogram(),
  maxPixels = 1e5,
  bestEffort = TRUE
)
hist <- hist$getInfo(); hist <- unlist(hist)
cat("Percent of map in each cluster:")

## Percent of map in each cluster:

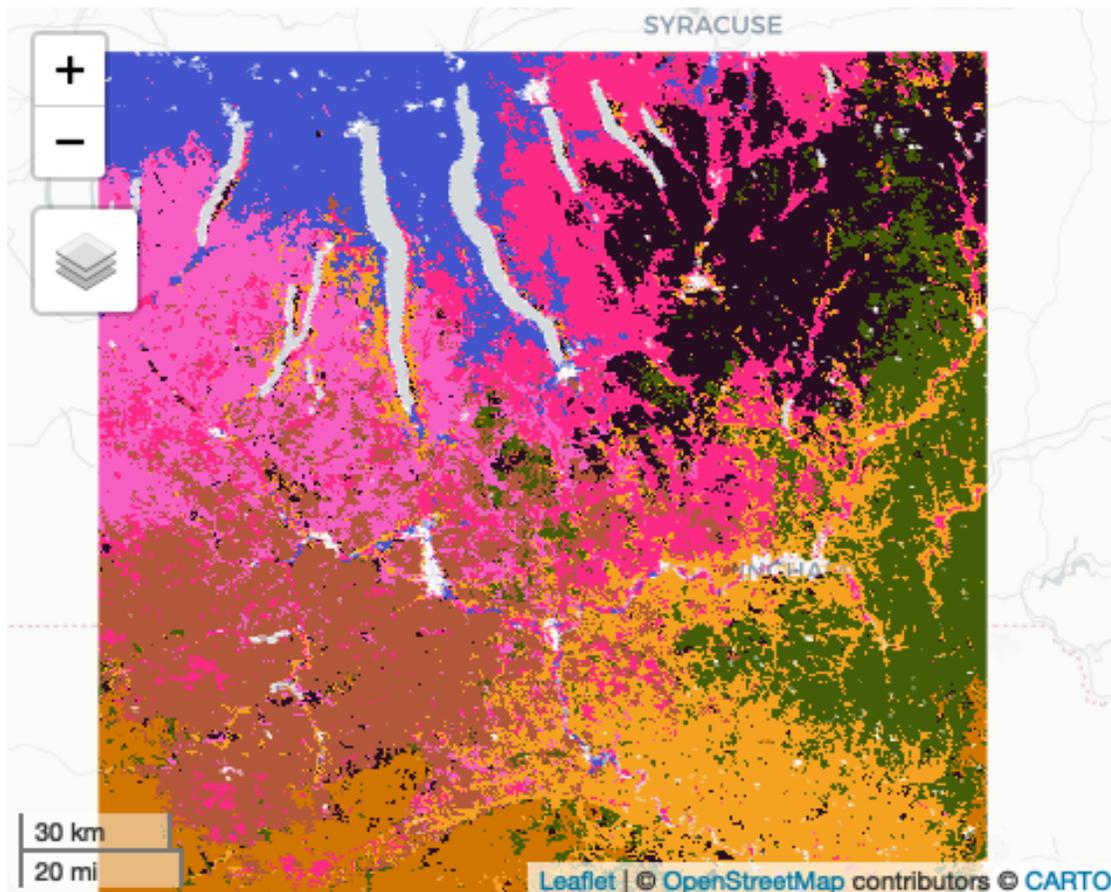
print(round(100*hist/sum(hist),1))
```

```
## cluster.0 cluster.1 cluster.2 cluster.3 cluster.4 cluster.5 cluster.6
cluster.7
##      14.6      11.8      16.7      16.5      12.2      11.5      7.0
9.6
```

In the example area the clusters are not well-balanced, but that is not to be expected with clusters of soil properties.

Display the map of the clusters, using the `randomVisualizer` method to assign colours to classes:

```
Map$centerObject(poi, zoom=8)
Map$addLayer(PCcluster$randomVisualizer())
```



Examine the map for reasonableness – this depends on the judgement of the soil scientist and on comparison with previous maps. Note that any spatial contiguity in the clustering is because of spatial contiguity in the PCs.

We can try different numbers of clusters, here left as an exercise.

Clustering – first 3 PCs

An alternative is to use just the first few PCs. These explain 67% of the variance, and form a knickpoint in the screeplot. This removes the influence of the less important PCs.

```

PCs3 <- PCs$select(0L:2L) # indexing starts from 0
training3 <- PCs3$sample(
  region = roi,
  scale = scale,
  tileSize = 16, # spread the computation over several processors
  numPixels = 10000L
)
clusters3 <- ee$Clusterer$
  wekaKMeans(nClusters=n_clusters, distanceFunction="Manhattan")$
  train(training3)
PCcluster3 <- PCs3$cluster(clusters3)
hist <- PCcluster3$reduceRegion(
  reducer = ee$Reducer$frequencyHistogram(),
  maxPixels = 1e5,
  bestEffort = TRUE
)
hist <- hist$getInfo(); hist <- unlist(hist)
cat("Percent of map in each cluster:")

## Percent of map in each cluster:

print(round(100*hist/sum(hist),1))

## cluster.0 cluster.1 cluster.2 cluster.3 cluster.4 cluster.5 cluster.6
cluster.7
##      9.0      17.6      16.7      9.3      17.8      14.0      8.7
6.9

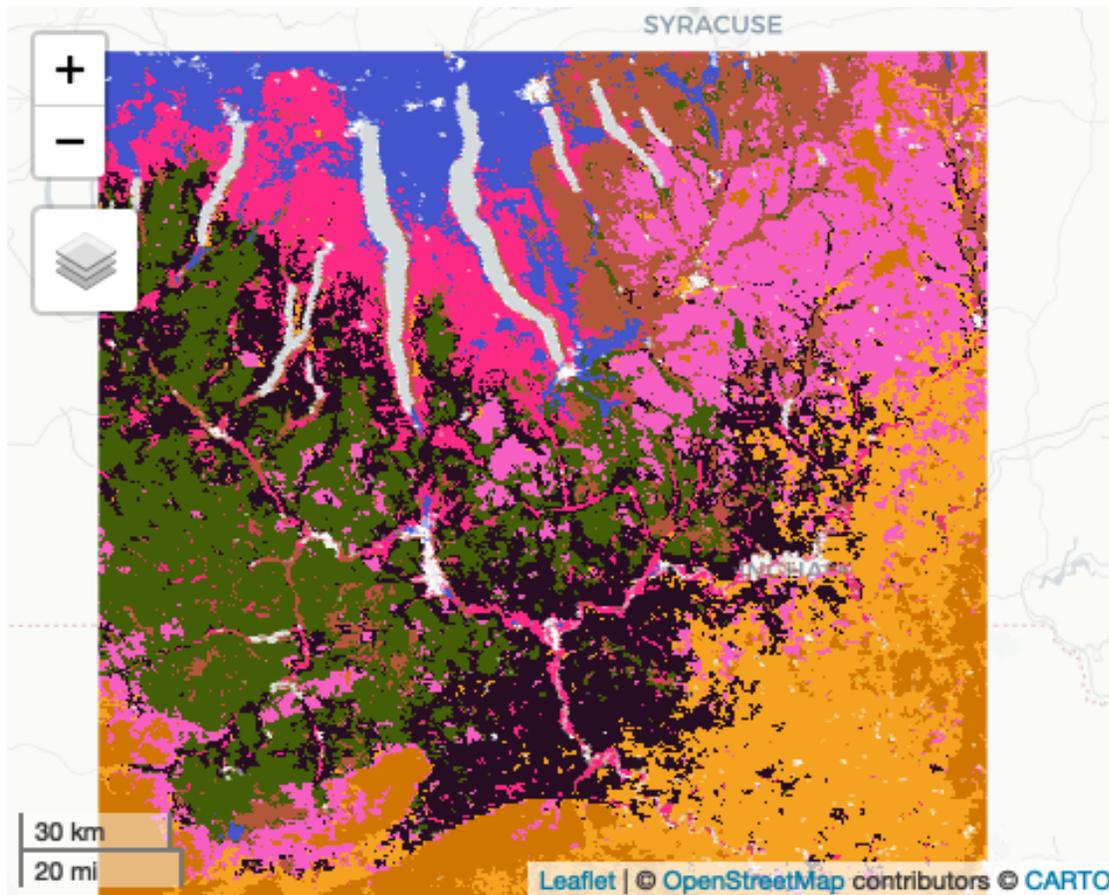
```

Again an unbalanced clustering.

```

Map$centerObject(poi, zoom=8)
Map$addLayer(PCcluster3$randomVisualizer())

```



For the example case, quite a different image than the clustering using all significant PCs.

Again, we can try different numbers of clusters; this is left as an exercise.

Exporting

Exporting can be to the user's "Assets" folder within GEE, or to the user's Google Drive. The latter can then be read in R, without using rgee. These exports are as "tasks" which are first set up and named and then started with the `start()` function applied to the named task.

Export is asynchronous and low-priority for GEE. The `ee_monitoring` function can be used to check the status of the task list. Generally this is done in the console.

For further processing in GEE

Each GEE user has an "Assets" folder. These are then available for use in other GEE scripts. We used ISRIC's "Assets" folder to [read the SoilGrids layers](#).

Export the PC image stack to your assets with the `ee_image_to_asset` function. Only export the "significant" PCs, i.e., those explaining 95% of the total variance of the 42 standardized layers.

```
task_pcs_asset <- ee_image_to_asset(PCs95,
                                   assetId = paste0(ee_user_info(quiet =
TRUE)$asset_home, "/PCs95"),
                                   overwrite = TRUE)
task_pcs_asset$start()
```

Export the clustered image based on all significant PCs.

```
task_clusters_asset <- ee_image_to_asset(PCcluster,
                                         assetId = paste0(ee_user_info(quiet =
TRUE)$asset_home, "/PCcluster"),
                                         overwrite = TRUE)
task_clusters_asset$start()
```

For further processing in R

R has a much richer set of algorithms and means to examine the results, in particular, to determine the optimum number of clusters. In R we can also do hierarchical clustering.

To do this, we need to get the images and/or training pixels into R. This is a GEE “task”, and is low priority for GEE. So we set up the task, start it, but do not wait for the task to complete. The completion can be checked with the `file.exists()` function, either interactively at the console, or within another script.

Only export the “significant” PCs, i.e., those explaining 95% of the total variance of the 42 standardized layers.

The export will be to the user’s Google Drive, under the listed folder, here `rgee`. If there are previous copies of the files to be exported, they will *not* be over-written.

```
task_pcs <- ee_image_to_drive(PCs95, folder="rgee",
                             timePrefix = FALSE,
                             fileNamePrefix = "PCs95",
                             fileFormat = "GeoTIFF")
task_pcs$start()
```

Export the training points:

```
task_train <- ee_table_to_drive(training, folder="rgee",
                                fileNamePrefix = "trainingPoints",
                                timePrefix = FALSE,
                                fileFormat = "GeoJSON")
task_train$start()
```

Export the clustered image derived from the first 3 PCs:

```
task_clusters <- ee_image_to_drive(PCcluster3, folder="rgee",
                                   fileNamePrefix = "PCcluster3",
                                   timePrefix = FALSE,
                                   fileFormat = "GeoTIFF")
task_clusters$start()
```