United Nations Development Programme

(UNDP)

Research in Geospatial Data Analytics and OpenAI for DiCRA Digital Public Good for Climate Resilience

Deliverable #3

REFERENCE: RFP-136-IND-2022

Date of Submission: 24-03-2023

Submitted by

ICRISAT
INTERNATIONAL CROPS RESEARCH INSTITUTE FOR THE SEMI-ARID TROPICS

GEE Code (JavaScript)

I. Detailed Methodology and Related Code

This study aimed at major remote-sensing products that capture important cropland characteristics:

Product 1 & 2: Croplands and non-croplands & Irrigated versus rainfed croplands

Product 3 & 4 & 5: Crop types, Length of Growing Periods and Crop intensity

Product 6: Crop stress

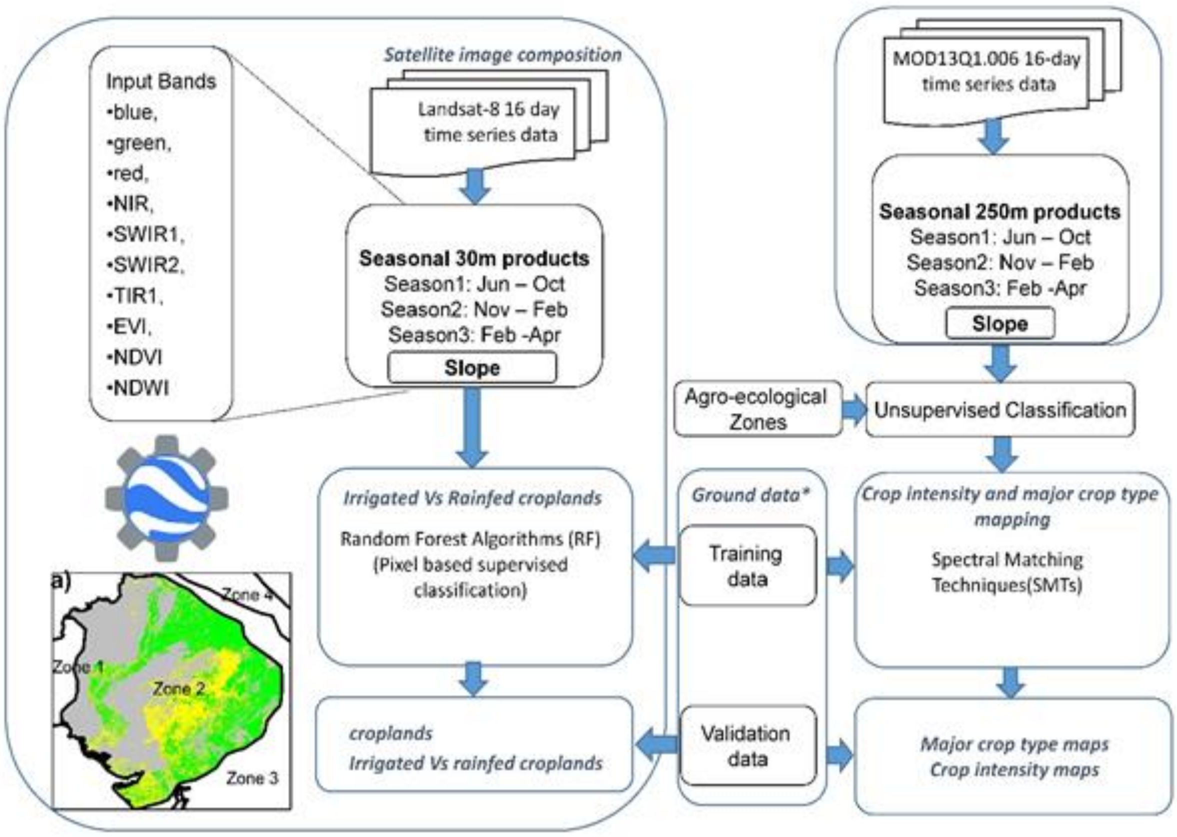

Figure 1: The methodology used for mapping cropland products

Methods for product 1 & product 2: Mapping croplands, irrigated and rainfed croplands using RF Machine Learning algorithm[1]

In making Product 1&2 to delineate croplands, irrigated croplands from rainfed croplands, we will adopt the RF machine learning algorithm and computing was performed on the GEE cloud platform, which

[1]Gumma, M.K., Thenkabail, P.S., Teluguntla, P.G., Oliphant, A., Xiong, J., Giri, C., Pyla, V., Dixit, S. and Whitbread, A.M., 2020. Agricultural cropland extent and areas of South Asia derived using Landsat satellite 30-m time-series big-data using random forest machine learning algorithms on the Google Earth Engine cloud. *GIScience & Remote Sensing*, *57*(3), pp.302-322.

is equipped with hitherto unheard-of petabyte-scale big data analytics. The RF machine learning algorithm is a pixel-based supervised classifier (Figure 1) The method involves the following steps:

- ***Reference training data collection***

The first step is to gather well-distributed reference training data used in developing irrigated versus rainfed cropland RF machine learning algorithm.

- ***Knowledge base creation and Random Forest Algorithm***

The knowledge base created was used to run the RF machine learning algorithms on the GEE cloud with multiyear seasonal data from data cube created (satellite image stack) to best separate irrigated areas from rainfed areas. The performance of the RF machine learning algorithms depends on the robustness of the reference training data. This requires large, well-distributed samples. At the same time, the quality of samples is very important. The process is iterative, meaning that after each run the classification accuracies are assessed. The process is repeated by refining the training data and improving the knowledge base till a highly accurate irrigated versus rainfed cropland product is obtained. Refinement of the knowledge base involves dropping and adding reference training data that have high degree of uncertainty until it ultimately leads to an optimal solution that has optimal producer's and user's accuracies.

With reference to above, here is the code

"

```
//var trainingPoint = ee.FeatureCollection('ft:1dqAd2OLYUNbj3ogdV71bqHqi4_aieiTFI30Yos_g');
//var train = [irrigated,rainfed,otehrlulc,water];
Var trainingPoint =
ee.FeatureCollection('ft:1Ox0UWr9HtSkKUt3jPQKzJe9xpdsVBWnFQIBN5FGy');
//Map.addLayer(trainingPoint,{palette["0000ff","00ff00",'ff0000'], min:[1],max:[3]});
//print (trainingPoint);
//throw('stop')


//var focal2 = image.focal_mode(35,'circle','meters',2);
```

```
//Map.addLayer(image, {palette:['00ff00','ff0000'], min:[1], max:[2], opacity:0.3, scale: 30});
//Map.addLayer(image3, {palette:['00ff00','ff0000'], min:1, max:3, opacity:0.3},
'Thailand');//,visPrams2;
////////////////////////////////////////////////////////////////////
//Some visualization parameters for stretching Landsat-like images
var vizParamsCO1 = {'min': 0.05,'max': [0.3,0.6,0.35],   'bands':'swir1,nir,red'};
var vizParamsCO2 = {'min': 0.15,'max': [0.35,0.8,0.4],   'bands':'swir1,nir,red', 'gamma': 1.6};
var vizParamsCO3 = {'min': 0.05,'max': [0.3,0.4,0.4],   'bands':'swir1,nir,red', 'gamma':1.6};
var vizParamsFalse = {'min': 0.1,'max': [0.3,0.3,0.3],   'bands':'nir,swir1,red'};
var vizParamsViz = {'min': 0.05, 'max': 0.3,'bands': 'red,green,blue', 'gamma': 1.6};
var vizParams = vizParamsCO1;
//////////////////////////////////////////////////////////////////////////
//Common band names
var bandNames = ee.List(['blue','green','red','nir','swir1','temp','swir2']);
var bandNumbers = [0,1,2,3,4,5,6];
///////////////////////////////////////////////////////////////
//Band combinations for each sensor corresponding to final selected corresponding bands
  var sensor_band_dict =ee.Dictionary({L8 : ee.List([1,2,3,4,5,9,6]),
                        L7 : ee.List([0,1,2,3,4,5,7])
  });
/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
//Helper function to not have to use EE list object
//Works al la range in Python
function range(start, stop, step){
  // start = parseInt(start);
  // stop = parseInt(stop);
    if (typeof stop=='undefined'){
        // one param defined
        stop = start;
        start = 0;
    }
    if (typeof step=='undefined'){
        step = 1;
    }
    if ((step>0 && start>=stop) || (step<0 && start<=stop)){
        return [];
    }
    var result = [];
    for (var i=start; step>0 ? i<stop : i>stop; i+=step){
        result.push(i);
    }
    return result;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Function to rescale image between 0 and 1
function rescale(img, exp, thresholds) {
    return img.expression(exp, {img: img})
        .subtract(thresholds[0]).divide(thresholds[1] - thresholds[0]);
  }
//////////////////////////////////////////////////////////////////
//Function to compute the "snowiness" of a pixel
//Uses most logic from the Google cloud score algorithm
//Calibrated in AK so may need calibration if used elsewhere
function snowScore(img){
      // Compute several indicators of snowyness and take the minimum of them.
      var score = ee.Image(1.0);
      // Snow is reasonably bright in the blue band.
      score = score.min(rescale(img, 'img.blue', [0.1, 0.3]));
        // Snow is reasonably bright in all visible bands.
      score = score.min(rescale(img, 'img.red + img.green + img.blue', [0.2, 0.8]));
        // // Excluded this for snow reasonably bright in all infrared bands.
      // score = score.min(
      //     rescale(img, 'img.nir + img.swir1 + img.swir2', [0.3, 0.8]));
        // Snow is reasonably cool in temperature.
      //Changed from [300,290] to [290,275] for AK
      score = score.min(rescale(img, 'img.temp', [300, 285]));
          // Snow is high in ndsi.
      var ndsi = img.normalizedDifference(['green', 'swir1']);
      ndsi = rescale(ndsi, 'img', [0.5, 0.7]);
      score = score.min(ndsi);
        return score.clamp(0,1)
          }
////////////////////////////////////////////////////
//Algorithm to compute liklihood of water
//Builds on logic from Google cloudScore algorithm
function waterScore(img){
      // Compute several indicators of water and take the minimum of them.
      var score = ee.Image(1.0);
          //Set up some params
      var darkBands = ['green','red','nir','swir2','swir1'];//,'nir','swir1','swir2'];
      var brightBand = 'blue';
      var shadowSumBands = ['nir','swir1','swir2'];
      //Water tends to be dark
```

```
      var sum = img.select(shadowSumBands).reduce(ee.Reducer.sum());
       sum = rescale(sum,'img',[0.35,0.2]).clamp(0,1)
      score = score.min(sum)

      //It also tends to be relatively bright in the blue band
      var mean = img.select(darkBands).reduce(ee.Reducer.mean());
      var std = img.select(darkBands).reduce(ee.Reducer.stdDev());
      var z = (img.select([brightBand]).subtract(std)).divide(mean)
      z = rescale(z,'img',[0,1]).clamp(0,1)
      score = score.min(z)

      // // Water is at or above freezing
      score = score.min(rescale(img, 'img.temp', [273, 275]));

      // // Water is nigh in ndsi (aka mndwi)
      var ndsi = img.normalizedDifference(['green', 'swir1']);
      ndsi = rescale(ndsi, 'img', [0.3, 0.8]);
            score = score.min(ndsi);
       return score.clamp(0,1)
            }
////////////////////////////////////////////////////////////////////////////////
//Helper functions for masking snow and water
//Thresholds between 0-1 where lower number masks more out
///////////////////////////////////////////////////////////////////////
function maskSnow(img){
  var ss = snowScore(img)
  return img.mask(img.mask().and(ss.lt(snowThresh)))
}
```

```
////////////////////////////////////////////////////////////////
function maskWater(img){
  var ws = waterScore(img)
  return img.mask(img.mask().and(ws.lt(waterThresh)))
}
///////////////////////////////////////////////////////////////
//Basic cloud busting function
function bustClouds(img){
    var t = img
    var cs = ee.Algorithms.Landsat.simpleCloudScore(img).select('cloud')
    var out = img.mask(img.mask().and(cs.lt(cloudThresh)))
    return out.copyProperties(t)
}
//////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
//Function to handle empty collections that will cause subsequent processes to fail
//If the collection is empty, will fill it with an empty image
function fillEmptyCollections(inCollection,dummyImage){
  var dummyCollection = ee.ImageCollection([dummyImage.mask(ee.Image(0))])
  var imageCount = inCollection.toList(1).length()
  return ee.ImageCollection(ee.Algorithms.If(imageCount.gt(0),inCollection,dummyCollection))
}
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//Function for adding common indices
function addIndices(in_image){
    in_image = in_image.addBands(in_image.normalizedDifference(['nir', 'red']).select([0],['NDVI']));
  in_image = in_image.addBands(in_image.normalizedDifference(['green',
'swir1']).select([0],['NDWI']));
  //EVI algorithm taken from: http://landsat.usgs.gov/documents/si_product_guide.pdf
  //EVI = (Band 4 – Band 3) / (Band 4 + 6 * Band 3 – 7.5 * Band 1 + 1)
  // var nir = in_image.select(['nir']).multiply(10000);
  // var red = in_image.select(['red']).multiply(10000);
  // var evi = (nir.subtract(red)).divide((nir))
  return in_image;
}
//////////////////////////////////////////////////////////////////////////////
//Adds the float year with julian proportion to image
function addDateBand(img){

 var d = ee.Date(img.get('system:time_start'))
 var y = d.get('year')
 d = y.add(d.getFraction('year'))
 var db = ee.Image.constant(d).select([0],['year']).float()
 return img.addBands(db)
 .copyProperties(img)
}
////////////////////////////////////////////////
// # Purpose: Remove the fringes of landsat 5 and 7 scenes.
// #
//Kernel for masking fringes found and L5 and L7 imagery
//////////////////////////////////////////////////
//Function to remove cloud shadows
//Works by identifying dark outliers in the infrared bands
function simpleTDOM(collection,zShadowThresh,zCloudThresh,maskAllDarkPixels){
  //Set up some variables
    var shadowSumBands = ['nir','swir1','swir2']
    var sSName = 'shadowSum'
    var startBandNames = ee.Image(collection.first()).bandNames();
       //Add the sum of the infrared to all images
    collection = collection.map(function(img){
     var shadowSum = img.select(shadowSumBands).reduce(ee.Reducer.sum()).select([0],[sSName])
     return img.addBands(shadowSum);
    })

    //If a first cut of masking very dark pixels is chosen, then mask all very dark pixels
    //This works kind of like the two-step approach in Fmask with individual image-based
    //masking and then a time series-based method
    if(maskAllDarkPixels === true){
     collection = collection.map(function(img){
       return img.mask(img.mask().and(img.select([sSName]).gt(shadowThresh)))
     })
    }
       //Compute the stdDev and mean of the sum of the infrared bands fo z-score analysis
    var shadowStd = collection.select(sSName).reduce(ee.Reducer.stdDev());
    var shadowMean = collection.select(sSName).mean();
       //Compute z-score and mask pixels falling below a specified threshold
    collection = collection.map(function(img){
     var tShadowSum =
img.select(shadowSumBands).reduce(ee.Reducer.sum()).select([0],['shadowSumT']);
     var zScore = tShadowSum.subtract(shadowMean).divide(shadowStd).select([0],['zShadow']);
     var m = zScore.gt(zShadowThresh).and(zScore.lt(zCloudThresh));
     return img.mask(img.mask().and(m)).select(startBandNames);
```

```
      })
      return collection
    }
/////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////
//Function for gathering, cloud masking, and cloud shadow masking Landsat imagery
function simpleCloudShadowFreeImages(startDate,endDate,startJulian,endJulian){

  //Get L7 images
  var l7 = ee.ImageCollection('LE7_L1T_TOA')
        .filterDate(startDate,endDate)
        .filter(ee.Filter.calendarRange(startJulian,endJulian))
        .filterBounds(studyArea)
        //May or may not want to run defringe on L7
        // if(runDefringe === true){
        //   print('Running defringe on L7')
        //   l7 = l7.map(defringeLandsat);
        // }
        //Bust clouds
        l7 = l7
        .map(bustClouds)
        .select(sensor_band_dict.get('L7'),bandNames)

  //Get L8 images and bust clouds
  var l8 = ee.ImageCollection('LC8_L1T_TOA')
        .filterDate(startDate,endDate)
        .filter(ee.Filter.calendarRange(startJulian,endJulian))
        .filterBounds(studyArea)
      .map(bustClouds)
      .select(sensor_band_dict.get('L8'),bandNames)

  //Merge collections
  var ls = ee.ImageCollection(l7.merge(l8));

  //Mask snow and/or water of elected
  if(shouldMaskSnow === true){
    print('Masking snow');
    ls = ls.map(maskSnow);
  }
  if(shouldMaskWater === true){
    print('Masking water');
    ls = ls.map(maskWater);
  }

  //Temporal Dark Outlier Mask (TDOM)
  //Z-score based cloud shadow removal
  if(runTDOM === true){
    ls =simpleTDOM(ls,zShadowThresh,zCloudThresh,maskAllDarkPixels)
  }
  ls = ls.map(addIndices);
  return ls
}
function exportLandsatComposite(composite,name,res,crs,noData){
  // var composite
=simpleCloudShadowFreeImages(startDate,endDate,startJulian,endJulian).median();
  addToMap(composite,vizParams,name)
  print(composite.bandNames());
  //var forExport = composite.select([0,1,2,3,4,6]).multiply(10000);//Multiply refl bands by 10k to
reduce to 16 bit
  //forExport = forExport.addBands(composite.select([5])).select([0,1,2,3,4,6,5]).int16();//Add thermal
back in and cast to 16 bit
  //output NDVI only
  var forExport = composite.select([0,1,2,3,4,5,6,7,8]).multiply(10000);//Multiply refl bands by 10k to
reduce to 16 bit
  var m = forExport.mask();//Get current null values
  forExport = forExport.mask(ee.Image(1));//Get rid of all null values
  forExport = forExport.where(m.not(),noData);//Reset null values to no data value
  Export.image(forExport,name,{'crs':crs,'region':regionJSON,'scale':res,'maxPixels':1e13})

  var exportViz = forExport.float().divide(10000).clip(studyArea).visualize(vizParams);
  addToMap(exportViz,{},name+'-8-bit',false)
  //Get URL for PNG
 //ar url =exportViz.getThumbURL({'dimensions':1500,'region':regionJSON,'format':'png'})
//print(url)
}
//////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////


//Parameters
var startDate = ee.Date.fromYMD(2014,6,1);
var endDate = ee.Date.fromYMD(2015,5,30);

var startJulian = 151;
var endJulian = 315;
var startJulian1 = 316;

var endJulian1 = 75;


var startJulian2 = 76;

var endJulian2 = 150;


var cloudThresh = 30;//Threshold for cloud masking (lower number masks more clouds)


var runTDOM = true;//Whether to run TDOM cloud shadow masking

var runDefringe = true;//Whether to run defringe algorithm on L5 and L7

var fringeCountThreshold = 279;//Define number of non null observations for pixel t

var runTDOM = true;//Whether to run TDOM

var maskAllDarkPixels = false;//Whether to perform a first cut masking of all dark pixels for shadow

var shadowThresh = 0.1;//If maskAllDarkPixels == true, a first cut of cloud shadow masking.

//Generally 0.1-0.15 works well

var zShadowThresh = -0.8;//Z-score threshold for cloud shadows. Generally -0.8 to -2.0 works well.

Masks more as it approaches 0

var zCloudThresh = 3;//Z-score threshold for masking any missed clouds.  Generally 3-4 works well

// var minNumberObservationsNeeded = 5;//Min number of observations needed for pixel to be

included in analysis

var shouldMaskSnow = false;//Whether to mask snow

var shouldMaskWater = false;//Whether to mask water

var waterThresh = 0.05;//Lower number masks more out  (0-1)

var snowThresh = 0.05;//Lower number masks more out (0-1)


var crs = 'EPSG:4326'; //Projection info for export

var outNoData = -99; //Null value on export
var outputName = 'SA_NDVI_151_315_14';//Descriptive name for export

var outputName1 = 'SA_NDVI_316_75_1415';

var outputName2 = 'SA_NDVI_76_150_15';

//Get all images and cloud and shadow mask them

var allImages = simpleCloudShadowFreeImages(startDate,endDate,startJulian,endJulian);

var allImages1 = simpleCloudShadowFreeImages(startDate,endDate,startJulian1,endJulian1);

var allImages2 = simpleCloudShadowFreeImages(startDate,endDate,startJulian2,endJulian2);


//Export the composite

//exportLandsatComposite(allImages.median(),outputName,30,crs,outNoData)


var ndviImage = ee.Image(allImages.mean(),outputName,30,crs,outNoData);

var ndviImage1 = ee.Image(allImages1.mean(),outputName,30,crs,outNoData);

var ndviImage2 = ee.Image(allImages2.mean(),outputName2,30,crs,outNoData);


//var vizParamsCO1 = {'min': 0.003,'max': [0.1,0.1,0.1],

'bands':'red_stdDev,green_stdDev,blue_stdDev', opacity:1};

var vizParamsCO1 = {'min': 0.05,'max': [0.4,0.4,0.35],   'bands':'red,green,blue'};

//Map.addLayer(ndviImage2, vizParamsCO1, 'stdev5');


var input = ee.Image.cat([ndviImage, ndviImage1, ndviImage2]);

//put = inputOrig.mask(image2.gt(1.5));

//var train = [thiCrop,thiNotcrop];

var TrainingSamples = ee.FeatureCollection(trainingPoint);

var training = input.sampleRegions({

   collection: TrainingSamples,

  properties: ['newCode'],

  scale: 30
});

Export.table(training, 'SA_GD_Training');

//print('training:',training);

//build classifier

var classifier = ee.Classifier.randomForest(200,0,1,0.5,true).train(training, 'newCode');

// print (classifier);

//classify the whole landscape

var classified = input.classify(classifier);

////Export the classified image

var jsonCoordString = studyArea.toGeoJSON();

Export.image(classified, 'SA_RF_2014_2367', {

 scale: 30,

 region: jsonCoordString,

 maxPixels:9999900000000

 });
“
```

Method for Product 3 & 4 & 5: Crop type[2], Length of Growing Periods and crop intensity mapping using quantitative spectral matching technique[3]

Satellite data was used to classify and identify crop types using quantitative spectral matching techniques (SMTs) (Figure 1). The SMTs involved developing ideal spectral signatures (ISSs), classifying images and obtaining class spectral signatures (CSSs), and matching class spectra with ideal spectra to identify and label crop type classes. Methodological steps involve the following steps:

A. Ideal spectral signatures.

Ideal spectral signatures were produced using NDVI time-series data with precise knowledge of croplands based on the unique reference samples obtained from the ground survey. The reference samples were classified according to their unique categories and grouped into homogeneous classes considering cropping intensity, crop types, and cropping system. The spectral signature curve explains crop behaviour over time. During the initial stage of crop growth, it has a smaller NDVI value; during peak growth, the NDVI value is high; and at harvest stage, the NDVI value is again smaller. Given the precise knowledge of the crop type based on ground data, we have clear understanding of how each crop behaves in terms of NDVI over time.

B. Class spectra generation.

Class spectra were generated based on unsupervised classification of MODIS 250 m, 16-day NDVI time-series data for the year 2014-2015 using the ISOCLASS cluster algorithm (ISODATA in ERDAS Imagine followed by progressive generalization. The initial classification was set at a maximum of 160 iterations and a convergence threshold of 0.99, which resulted in 160 classes for the study area. Spectral signatures were generated for every individual class.

C. Matching of class spectra on the basis of ideal spectra to group classes using SMTs.

The process of SMT mainly involves two steps:

Grouping similar class spectra: The initial 160 classes obtained from unsupervised classification (called class spectra) were grouped into a sub groups based on quantitative spectral matching techniques (QSMTs), i.e., classes nearly similar spectral profiles.

[2] Gumma, M.K., Thenkabail, P.S., Panjala, P., Teluguntla, P., Yamano, T. and Mohammed, I., 2022. Multiple agricultural cropland products of South Asia developed using Landsat-8 30 m and MODIS 250 m data using machine learning on the Google Earth Engine (GEE) cloud and spectral matching techniques (SMTs) in support of food and water security. *GIScience & Remote Sensing*, *59*(1), pp.1048-1077.

[3] Gumma, M.K., Thenkabail, P.S., Teluguntla, P., Rao, M.N., Mohammed, I.A. and Whitbread, A.M., 2016. Mapping rice-fallow cropland areas for short-season grain legumes intensification in South Asia using MODIS 250 m time-series data. *International Journal of Digital Earth*, *9*(10), pp.981-1003.

Matching group of similar classes with ideal spectra: Spectral matching involves, finding some ideal spectra that matches with group of class spectra. This is achieved by group of quantitative spectral matching techniques (QSMTs) methods such as spectral correlation similarity (SCS) R-square values between the ideal spectra and class spectra. When an R-square value of 0.80 or higher is achieved in correlation between ideal spectra and class spectra then it is considered as a match.

Thus how the crop products like crop extent, crop type, length of growing periods and crop intensity will be mapped based on the above methodology.

Product 6: Crop stress Monitoring[4]

Crop stress is one inevitable phenomenon which looms over all aspects of human life and affects crop production in Rainfed areas. Assessing crop stress using geospatial datasets in the public domain, such as Normalized Difference Vegetation Index (NDVI) and Land and Surface Water Index (LSWI) product derived from the Moderate Resolution Imaging Spectroradiometer (MODIS), helps insurance agencies to understand different stresses in croplands.

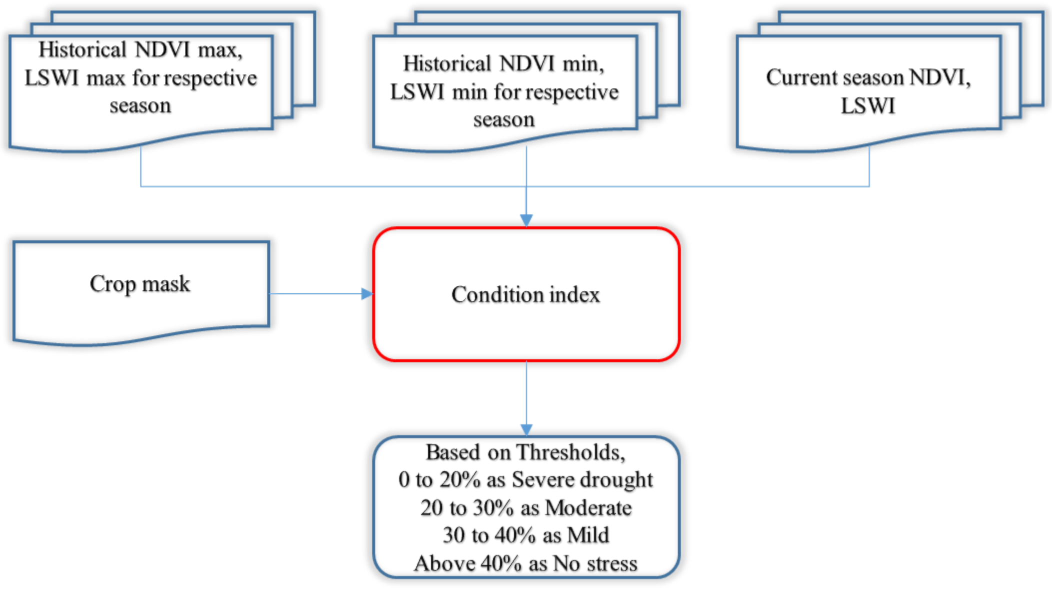
Historical NDVI max, LSWI max for respective season
Historical NDVI min, LSWI min for respective season
Current season NDVI, LSWI
Crop mask
Condition index
Based on Thresholds,
0 to 20% as Severe drought
20 to 30% as Moderate
30 to 40% as Mild
Above 40% as No stress

Figure 2: Methodology of Crop stress monitoring

We use MODIS NDVI and LSWI product to map crop stress areas and changes in cropland areas due to rainfall. A methodology was developed to identify crop stress-affected areas using the MODIS-based NDVI, LSWI and compare these crop areas with those during a normal year (Figure 2). A long-term average of NDVI during the rainy (Kharif) season (June-October) will compared to a current crop year. Rainy season NDVI and crop area changes were identified. Secondary data will use to support the crop stress analysis. Spectral matching techniques were used to categorize the crop stress affected cropland

[4] Gumma, M.K., Nelson, A. and Yamano, T., 2019. Mapping drought-induced changes in rice area in India. *International journal of remote sensing*, *40*(21), pp.8146-8173.

areas into three classes -- severe, moderate and mild -- based on the intensity of damage assessed through field sampling. Spectral signatures were generated based on these ground survey samples.

Attached working Procedure as separate Files

1. Pre-requisite for understanding GEE code
2. GEE code (Common Code) for Product 1 & 2
3. Working Procedures of Product 3&4&5
4. Working Procedure of Product 6
5. Ground data (Telangana, Odisha & Jharkhand) available