# KNOWLEDGE EXTRACTION IN AGRICULTURE USING MACHINE LEARNING ALGORITHMS

*A thesis submitted to*

*Indian Institute of Science, Bangalore*

*for the award of the degree*

*of*

**Master of Technology**

**in Civil Engineering with Specialization**

**in "Water Resources and Environmental Engineering"**

*by*

**Rukmangadan D**
SR No. 05-05-00-10-42-16-1-13305

Under the guidance of
**Prof. M. Sekhar**


**DEPARTMENT OF CIVIL ENGINEERING**

**INDIAN INSTITUTE OF SCIENCE, BANGALORE**

**JUNE 2018**

**DECLARATION**

I certify that

a. the work contained in this report is original and has been done by me under the guidance of my supervisor.
b. the work has not been submitted to any other Institute for any degree or diploma.
c. I have followed the guidelines provided by the Institute in preparing the report.
d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.
f. I hereby declare that I am the sole author of this thesis. I authorize Indian Institute of Science to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature of the Student

# CERTIFICATE

This is to certify that the Dissertation Report entitled, "**KNOWLEDGE EXTRACTION IN AGRICULTURE USING MACHINE LEARNING ALGORITHMS**" submitted by Mr. "Rukmangadan D" to Indian Institute of Science, Bangalore, India, is a record of bonafide Project work carried out by him/her under my/our supervision and guidance and is worthy of consideration for the award of the degree of Master of Technology in Civil Engineering with Specialization in "Water Resources and Environmental Engineering" of the Institute.

_____

Supervisor

Date:

## ACKNOWLEDGEMENTS

I would like to take this opportunity to express my deep sense of gratitude and profound feeling of admiration to my thesis supervisor. Many thanks to all those who helped me in this work.

**CONTENTS**

# Chapter 1

## Introduction

Machine Learning algorithms are being widely used nowadays in all fields of science wherever huge amount of data is generated and agriculture is also not an exception for this. Farmers in many parts of India are largely dependent on timely rainfall for harvest and subsequent profits. Uncertainty surrounding this phenomenon has also haunted them since the beginning of civilization. Over time however, this uncertainty had reduced significantly as farmers back in the day could almost accurately plant crops based on previous experience with weather conditions. This wisdom has been passed on from one generation of farmers to the other. Gradual onset of global warming and climate changes over the last century has slowly-yet steadily put this wisdom out of use. As for rain-fed farmers preparing for agriculture, soil-water equation is fragile and any delay in rainfall could easily destroy the harvest. When age old systems fail, look to the future.

One of the fastest growing areas under the discipline of "artificial intelligence" is machine learning. And this technology is being deployed across modern agriculture to create solutions with greater accuracy and at unprecedented scale. Of the millions of combinations, advanced software greatly narrows the search. Machine learning can be used draw conclusions from various sets of raw data. Researchers in modern agriculture are testing their theories at greater scale and helping make more accurate, real-time predictions. Modern agriculture has the potential to discover even more ways to conserve water, use nutrients and energy more efficiently, and adapt to climate change. With the advent of technology, numerous advancements have taken place so that now we are in a place where we can measure field soil moisture using Remote Sensing satellites. The main goal in this study is to derive some meaningful relation from the variation between relative soil moisture obtained from the satellite data and field soil moisture obtained from 'Berambadi' Region.

# Chapter 2

## Literature Review

Though we can rely on moisture data given by satellite to a certain extent, we cannot rely on it completely as it is affected by many factors and it may not give the actual soil moisture. H. McNairna,*, C. Duguayb, B. Briscoc, T.J. Pultza examined the effect of soil and crop residue characteristics on polarimetric radar response. This study examines the sensitivity of linear polarizations and polarimetric parameters to conditions present on agricultural fields during the period of preplanting and postharvest. The co-polarizations signature plots are also discussed. Results indicate that the dominant scattering mechanism from these fields varies depending on the type and amount of residue cover, and whether the crop had been harvested. Radar parameters most sensitive to volume and multiple scattering perform best at characterizing these surface conditions. The scattering mechanisms associated with standing senesced vegetation, no-till fields, and tilled fields varied. Double-bounce, multiple, and volume scattering were all present in standing vegetation, while for no-till fields multiple scattering dominated. The pedestal height was also unique for each of these classes, with larger pedestals associated with standing crops and no-till fields. This confirms the sensitivity of pedestal height to multiple and volume scattering.

Jun Wen, Zhongbo Su examined that the radar backscattering coefficient is mainly determined by surface soil moisture, vegetation and land surface roughness under a given configuration of the satellite sensor. It is observed that the temporal variations of the three variables are different, the variation of vegetation and roughness are at the longer temporal scales corresponding to climate and cultivation practices, while soil moisture varies at a shorter temporal scale in response to weather forcing. Relative soil moisture is a function of field soil moisture and saturation capacity of the soil. The results show that the estimated relative soil moisture corresponds closely to vegetation and land surface roughness.

B.J. Choudury, T.J. Schmugge, R.W. Newton and A.Chang studied the effect of surface roughness on the brightness temperature of a moist terrain through the modification of Fresnel reflection coefficient and using the radiative transfer equation. The modification involves introduction of a single parameter to characterize the roughness. It is shown that this parameter depends on both

the surface height variance and the horizontal scale of the roughness. Model calculations are in good quantitative agreement with the observed dependence of the brightness temperature on the moisture content in the surface layer.

J.R. Wang, P.E. O'Neill, T.J. Jackson, E.T. Engman conducted experiment on remote sensing of soil moisture content was conducted over bare fields with microwave radiometers at the frequencies of 1.4 GHz, 5 GHz, and 10.7 GHz. Three bare fields with different surface roughnesses and soil textures were prepared for the experiment. Ground truth acquisition of soil temperatures and moisture contents for 5 layers down to the depths of 15 cm was made concurrently with radiometric measurements. The experimental results show that the effect of surface roughness is to increase the soils' brightness temperature and to reduce the slope of regression between brightness temperature and moisture content. The slopes of regression for soils with different textures are found to be comparable, and the effect of soil texture is reflected in the difference of regression line intercepts at brightness temperature axis.

L. Breiman Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy. The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Thus, it is seen that the relative soil moisture determined by the satellites cannot be blindly correlated with the field soil moisture measured manually even though the saturation capacity of the soil is known. The main factors that lie in between the correlation of relative soil moisture and field soil moisture are the type of crop, crop coverage or crop growth basically how much the crop has covered the field at a given instance and also to the soil class based on texture and type of soil to which it belongs.

# Chapter 3

## Approach and Methodology

The field soil moisture data is obtained from 112 sites, from each site manually. These are the sites where the surface soil moisture data is available for year 2016 and 2017.


Image Showing 112 sites where soil moisture is measured routinely

Out of the 112 sites, 92 sites fall on the Berambadi region of Gundlupet Taluk and 20 sites fall in Vaddagare region of Koratagere Taluk.

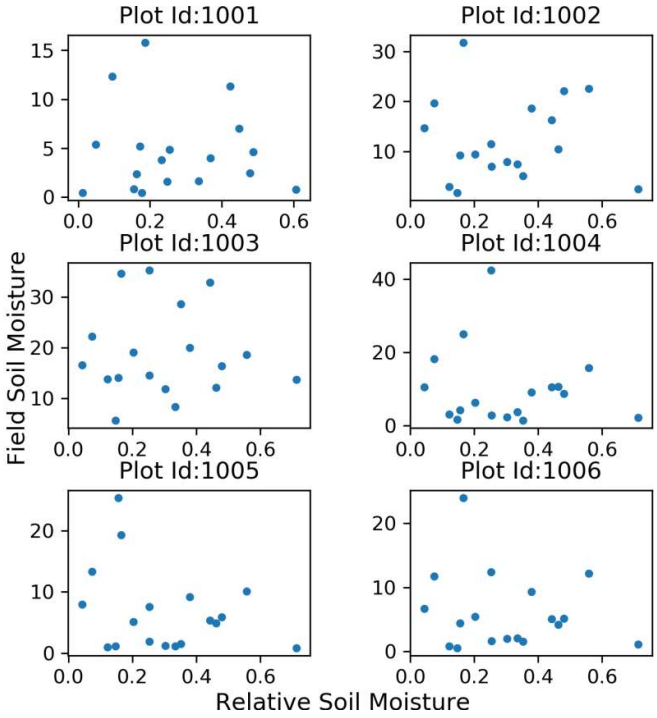
Each site is given a Plot ID for analysis purpose. The GPS coordinates of all the 112 sites are listed in the table below:

| Plot_Id | Latitude | Longitude | Plot_Id | Latitude | Longitude |
|---|---|---|---|---|---|
| 1001 | 11.79922 | 76.668 | 1057 | 11.78055 | 76.55742 |
| 1002 | 11.79856 | 76.65578 | 1058 | 11.7807 | 76.55765 |
| 1003 | 11.79863 | 76.65544 | 1059 | 11.78117 | 76.55781 |
| 1004 | 11.79923 | 76.65604 | 1060 | 11.78136 | 76.55815 |
| 1005 | 11.79984 | 76.656 | 1061 | 11.78165 | 76.56029 |
| 1006 | 11.79986 | 76.65602 | 1062 | 11.78109 | 76.56032 |
| 1007 | 11.79728 | 76.65138 | 1063 | 11.77904 | 76.55967 |
| 1008 | 11.79809 | 76.65148 | 1064 | 11.77926 | 76.55979 |
| 1009 | 11.79844 | 76.65142 | 1065 | 11.7787 | 76.56049 |
| 1010 | 11.79927 | 76.65085 | 1066 | 11.77832 | 76.56054 |
| 1011 | 11.787 | 76.63294 | 1067 | 11.77797 | 76.56057 |
| 1012 | 11.78756 | 76.63284 | 1068 | 11.77771 | 76.5606 |
| 1013 | 11.78752 | 76.6326 | 1069 | 11.77713 | 76.56074 |
| 1014 | 11.7875 | 76.63216 | 1070 | 11.74682 | 76.59141 |
| 1015 | 11.78108 | 76.62736 | 1071 | 11.74689 | 76.5917 |

| | | | | | |
|---|---|---|---|---|---|
| 1016 | 11.77725 | 76.62684 | 1072 | 11.74646 | 76.59168 |
| 1017 | 11.7624 | 76.62413 | 1073 | 11.74611 | 76.59158 |
| 1018 | 11.76174 | 76.62406 | 1074 | 11.74571 | 76.59169 |
| 1019 | 11.76173 | 76.62377 | 1075 | 11.74298 | 76.59072 |
| 1020 | 11.76173 | 76.62354 | 1076 | 11.74327 | 76.59087 |
| 1021 | 11.75631 | 76.60838 | 1077 | 11.74382 | 76.59097 |
| 1022 | 11.75692 | 76.60808 | 1078 | 11.74978 | 76.58994 |
| 1023 | 11.75693 | 76.60837 | 1079 | 11.74981 | 76.59027 |
| 1024 | 11.75613 | 76.59836 | 1080 | 11.74996 | 76.5918 |
| 1025 | 11.76326 | 76.59971 | 1081 | 11.74967 | 76.59184 |
| 1026 | 11.76336 | 76.59979 | 1082 | 11.72205 | 76.77913 |
| 1027 | 11.75485 | 76.58993 | 1083 | 11.77626 | 76.56072 |
| 1028 | 11.75639 | 76.58618 | 1084 | 11.73057 | 76.78211 |
| 1029 | 11.75696 | 76.58621 | 1085 | 11.73088 | 76.78255 |
| 1030 | 11.75735 | 76.58667 | 1086 | 11.73107 | 76.78261 |
| 1031 | 11.76143 | 76.58652 | 1087 | 11.73156 | 76.78283 |
| 1032 | 11.76101 | 76.58652 | 1088 | 11.7305 | 76.78417 |
| 1033 | 11.76069 | 76.58655 | 1089 | 11.73321 | 76.79298 |
| 1034 | 11.76132 | 76.58789 | 1090 | 11.73319 | 76.79266 |
| 1035 | 11.76166 | 76.58801 | 1091 | 11.73318 | 76.7923 |
| 1036 | 11.76118 | 76.58812 | 1092 | 11.73217 | 76.78549 |
| 1037 | 11.7607 | 76.58531 | 1093 | 11.73271 | 76.78565 |
| 1038 | 11.76074 | 76.58531 | 1094 | 11.73299 | 76.78532 |
| 1039 | 11.76127 | 76.58556 | 1095 | 11.73435 | 76.78511 |
| 1040 | 11.76129 | 76.5858 | 1096 | 11.73442 | 76.78544 |
| 1041 | 11.76101 | 76.58595 | 1097 | 11.73601 | 76.78595 |
| 1042 | 11.76172 | 76.57917 | 1098 | 11.73629 | 76.78589 |
| 1043 | 11.76088 | 76.57921 | 1099 | 11.73549 | 76.78456 |
| 1044 | 11.76086 | 76.57944 | 1100 | 11.7357 | 76.78471 |
| 1045 | 11.76213 | 76.57811 | 1101 | 11.73586 | 76.78471 |
| 1046 | 11.76885 | 76.56762 | 1102 | 11.57372 | 76.78369 |
| 1047 | 11.769 | 76.56771 | 1103 | 11.78131 | 76.62754 |
| 1048 | 11.77918 | 76.58606 | 1104 | 11.76464 | 76.62472 |
| 1049 | 11.77825 | 76.58636 | 1105 | 11.76534 | 76.62481 |
| 1050 | 11.77766 | 76.58646 | 1106 | 11.76518 | 76.62397 |
| 1051 | 11.77808 | 76.58685 | 1107 | 11.76144 | 76.62363 |
| 1052 | 11.77931 | 76.60627 | 1108 | 11.75825 | 76.5869 |
| 1053 | 11.77863 | 76.60614 | 1109 | 11.758 | 76.58597 |
| 1054 | 11.77993 | 76.55673 | 1110 | 11.76232 | 76.57731 |
| 1055 | 11.7799 | 76.55658 | 1111 | 11.77346 | 76.56834 |
| 1056 | 11.78011 | 76.55695 | 1112 | 11.77403 | 76.56823 |

While reviewing the literature, it is noted that relative soil moisture is a function of Field Soil Moisture (FSM) and saturation capacity of the soil. Hence, at a particular site, the relative soil moisture should have given a proper correlation with field soil moisture, but it has not.

Here, 'Relative Soil Moisture' (RSM) refers to the soil moisture data obtained through satellite which is provided by VATI project of 'Aapah innovations'. 'Field Soil Moisture' (FSM) refers to soil moisture measured from the site manually.




Plot showing Field Soil Moisture vs Relative Soil Moisture (for 6 sites)

But, when scatter plots of 'relative soil moisture' vs 'field soil moisture' are made, it does not show a good correlation. And, this leads to further thought why such variation is shown.

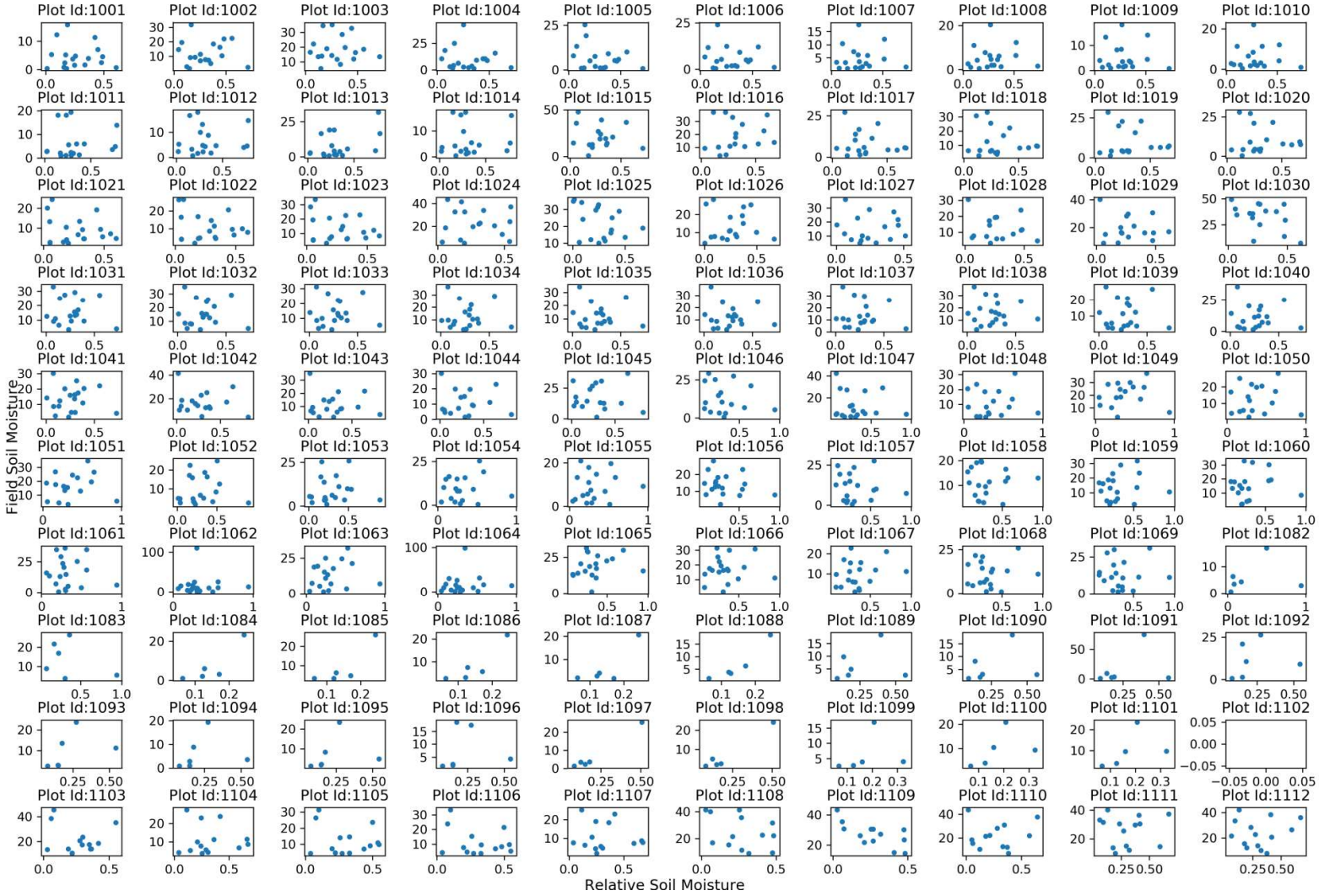
| **Plot_ID** | Correlation | **Plot_ID** | Correlation | **Plot_ID** | Correlation | **Plot_ID** | Correlation |
|---|---|---|---|---|---|---|---|
| **1001** | -0.066 | **1029** | -0.099 | **1057** | -0.092 | **1085** | 0.850 |
| **1002** | -0.004 | **1030** | -0.468 | **1058** | -0.016 | **1086** | 0.881 |
| **1003** | -0.030 | **1031** | -0.088 | **1059** | 0.135 | **1087** | 0.816 |
| **1004** | -0.127 | **1032** | 0.009 | **1060** | 0.101 | **1088** | 0.936 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1005** | -0.308 | **1033** | 0.010 | **1061** | -0.057 | **1089** | 0.223 |
| **1006** | -0.169 | **1034** | -0.032 | **1062** | -0.007 | **1090** | 0.300 |
| **1007** | 0.038 | **1035** | -0.071 | **1063** | 0.176 | **1091** | 0.342 |
| **1008** | 0.059 | **1036** | -0.075 | **1064** | 0.082 | **1092** | 0.194 |
| **1009** | -0.016 | **1037** | -0.084 | **1065** | 0.197 | **1093** | 0.438 |
| **1010** | 0.016 | **1038** | -0.036 | **1066** | 0.088 | **1094** | 0.190 |
| **1011** | 0.088 | **1039** | 0.014 | **1067** | 0.238 | **1095** | 0.232 |
| **1012** | 0.082 | **1040** | -0.058 | **1068** | 0.110 | **1096** | 0.092 |
| **1013** | 0.456 | **1041** | -0.023 | **1069** | 0.030 | **1097** | 0.987 |
| **1014** | 0.103 | **1042** | -0.178 | **1070** | | **1098** | 0.973 |
| **1015** | -0.142 | **1043** | -0.104 | **1071** | | **1099** | 0.262 |
| **1016** | 0.145 | **1044** | -0.042 | **1072** | | **1100** | 0.479 |
| **1017** | -0.174 | **1045** | -0.058 | **1073** | | **1101** | 0.465 |
| **1018** | -0.131 | **1046** | -0.156 | **1074** | | **1102** | |
| **1019** | -0.089 | **1047** | -0.115 | **1075** | | **1103** | -0.138 |
| **1020** | -0.063 | **1048** | 0.028 | **1076** | | **1104** | -0.040 |
| **1021** | -0.182 | **1049** | 0.090 | **1077** | | **1105** | -0.190 |
| **1022** | -0.209 | **1050** | 0.043 | **1078** | | **1106** | -0.282 |
| **1023** | -0.239 | **1051** | 0.144 | **1079** | | **1107** | -0.093 |
| **1024** | -0.058 | **1052** | 0.083 | **1080** | | **1108** | -0.355 |
| **1025** | -0.357 | **1053** | 0.072 | **1081** | | **1109** | -0.700 |
| **1026** | -0.009 | **1054** | 0.109 | **1082** | 0.217 | **1110** | 0.068 |
| **1027** | -0.023 | **1055** | 0.131 | **1083** | -0.338 | **1111** | 0.007 |
| **1028** | -0.172 | **1056** | -0.217 | **1084** | 0.860 | **1112** | 0.096 |

Table showing correlation between ‘relative soil moisture’ and ‘field soil moisture’ for each site

From the correlation table, it is seen that most of the sites have very poor correlation, mostly having $R^2$ values of less than 0.1. Thus, to bring a good result, just comparing between these two is not sufficient, but it is required to bring other parameters which cause this problem.

It is seen in the literature review that relative soil moisture measured using satellite gets affected by various factors such as the type of crop, crop coverage or crop growth basically how much the crop has covered the field at a given instance and also to the soil class based on texture and type of soil to which it belongs.




Field Soil Moisture vs Relative Soil Moisture Scatter Plot (for all 112 sites)

The compiled data used for understanding the relation between relative soil moisture data obtained from satellite and field soil moisture consists of crop type and soil class. The data needed for analysis is defined and organized in a manner to feed into machine learning algorithms.

The software utilized for extracting and organizing data are (i) Quantum GIS, commonly called as QGIS – an open source software, extended its capability with "PyQGIS", (ii) Geospatial Data Abstraction Library, in short "GDAL", for making pixel level calculations on the satellite data obtained and the programming language used is Python.

Crop Type:

The type of crop that is grown in farmer's field is available along with date. Frequency of data available is approximately twice a month for year 2016 and 2017. Total number of types of crop available from all 112 sites is 51.

| No | Type | No | Type | No | Type |
|---|---|---|---|---|---|
| 1 | No_Crop | 18 | Sunflower | 35 | Potato |
| 2 | Maize | 19 | Sugarcane | 36 | Maize+CountryBean |
| 3 | Chickpea | 20 | Garlic | 37 | CountryBean+Maize |
| 4 | Mariegold | 21 | Beetroot+Banana | 38 | Turmeric+Onion |
| 5 | Ragi | 22 | Banana | 39 | Pumkin |
| 6 | Groundnut+Marieg old | 23 | Watermelon+Banana+PolythynPla stic | 40 | HorseGram+CountryB ean |
| 7 | Cotton | 24 | Banana+Watermelon | 41 | CountryBean+Sunflow er |
| 8 | FieldBean | 25 | Ginger | 42 | Onion+Turmeric+Bana na |
| 9 | CountryBean | 26 | Weed | 43 | Watermelon |
| 10 | Sunflower+Maize | 27 | Cabbage | 44 | Banana+Beetroot |
| 11 | Sorghum | 28 | Garlic+Cabbage | 45 | Sunflower+Sorghum |
| 12 | HorseGram | 29 | Beetroot | 46 | Maize+Beetroot |
| 13 | Onion+Turmeric | 30 | Onion | 47 | Onion+Beans |
| 14 | Turmeric | 31 | Maize+Mariegold | 48 | Chilly+Beans |
| 15 | Groundnut | 32 | Toor+Chilly | 49 | Sorghum+Pulses |
| 16 | Beans | 33 | Chilly+Turmeric | 50 | Onion+Toor |
| 17 | Tomato | 34 | Chilly | 51 | Toor |

Since the number of classes are way too many, it is decided to reduce the number of soil classes based on plant leaf coverage area and its height of growth. Keeping this in mind, the plants are divided into 9 classes.

| | Less coverage | Medium coverage | Dense coverage |
|---|---|---|---|
| Short | 1 | 2 | 3 |
| Medium | 4 | 5 | 6 |
| Tall | 7 | 8 | 9 |

| Crop_Type | Crop Class | Crop_Type | Crop Class | Crop_Type | Crop Class |
|---|---|---|---|---|---|
| **No_Crop** | 1 | Onion | 2 | CountryBean | 6 |
| **Weed** | 1 | Toor+Chilly | 2 | HorseGram+CountryBean | 6 |
| **Pumkin** | 1 | Chilly+Turmeric | 2 | CountryBean+Sunflower | 6 |
| **Watermelon** | 1 | Chilly | 2 | Sunflower+Maize | 7 |
| **Ragi** | 2 | Potato | 2 | Sorghum | 7 |
| **Cotton** | 2 | Turmeric+Onion | 2 | Sunflower | 7 |
| **FieldBean** | 2 | Onion+Beans | 2 | Maize+CountryBean | 8 |
| **HorseGram** | 2 | Chilly+Beans | 2 | CountryBean+Maize | 8 |
| **Onion+Turmeric** | 2 | Onion+Toor | 2 | Sunflower+Sorghum | 8 |
| **Turmeric** | 2 | Toor | 2 | Sorghum+Pulses | 8 |
| **Groundnut** | 2 | Beetroot | 3 | Sugarcane | 9 |
| **Beans** | 2 | Mariegold | 4 | Beetroot+Banana | 9 |
| **Tomato** | 2 | Maize+Mariegold | 4 | Banana | 9 |
| **Garlic** | 2 | Maize | 5 | Watermelon+Banana+PolythynPlastic | 9 |
| **Ginger** | 2 | Groundnut+Marigold | 5 | Banana+Watermelon | 9 |
| **Cabbage** | 2 | Maize+Beetroot | 5 | Onion+Turmeric+Banana | 9 |
| **Garlic+Cabbage** | 2 | Chickpea | 6 | Banana+Beetroot | 9 |

Relative Soil Moisture:

The relative soil moisture is obtained with respect to GPS coordinates of sites from the moisture data files which are present in 'GeoTiff' format. The tool used for extracting relative soil moisture from 'GeoTiff' files is 'Geospatial Data Abstraction Library' available in the PyQGIS console. The code snippet used is as follows:

Code Snippet

```
import pickle
import numpy as np
import pandas as pd
import os, os.path
from datetime import datetime, timedelta
```

```
from pandas import ExcelWriter
from openpyxl import load_workbook
from osgeo import gdal, osr
from qgis.core import QgsRasterLayer

basepath = 'D:\MASTERS-PROJECT\\'
subBasepath = basepath + 'MTECH-PROJ-Prob_1_VATI_vs_Field_Moisture\\'
in_file = subBasepath + 'Soil_Moisture_Berambadi_1617_CropAge_out.xlsx'
out_file = subBasepath + 'Soil_Moisture_Berambadi_1617_CropAge_VATI_out.xlsx'

sm_folder = 'D:\MASTERS-PROJECT\sm_VATI\\'

SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')

SM_ML_Dates_df = SM_ML_df[['Date']]
SM_ML_Dates_df.drop_duplicates(subset=['Date'], keep='first', inplace=True)
SM_ML_Dates_df.reset_index(drop=True, inplace=True)

# print (SM_ML_Dates_df)

sm_VATI_LatLon_DF = SM_ML_df[['Plot_Id', 'Latitude', 'Longitude']]

sm_VATI_LatLon_DF.drop_duplicates(subset=['Plot_Id'], keep='first', inplace=True)
sm_VATI_LatLon_DF.reset_index(drop=True, inplace=True)

for index, SM_ML_Dates_df_row in SM_ML_Dates_df.iterrows():
    date_dfval = SM_ML_Dates_df_row['Date']
    date = str(date_dfval)
    date = datetime.strptime(date, '%Y-%m-%d %H:%M:%S')
    date = date.strftime('%Y%m%d')
    file_fullPath = sm_folder + date + '.tif'
    if(os.path.isfile(file_fullPath)):
        # print (date)
        raster_ds = gdal.Open(file_fullPath)
        rlayer = QgsRasterLayer(file_fullPath)

        geoTrans = raster_ds.GetGeoTransform()
        ulX = geoTrans[0]
        ulY = geoTrans[3]
        pxUnitX = geoTrans[1]
        pxUnitY = geoTrans[5] # this value is -ve

        prj = raster_ds.GetProjectionRef()
        srs = osr.SpatialReference(wkt=prj)
        EPSG_num = srs.GetAttrValue('authority', 1)

        crsSrc = QgsCoordinateReferenceSystem(4326)      # WGS 84
        crsDest = QgsCoordinateReferenceSystem(int(EPSG_num))   # WGS 84 / UTM zone 43N
        xform = QgsCoordinateTransform(crsSrc, crsDest)
```

```
    sm_VATI_nparray = np.array([])
    for index, sm_VATI_LatLon_DF_row in sm_VATI_LatLon_DF.iterrows():
        pt_gps = QgsPoint(float(sm_VATI_LatLon_DF_row['Longitude']),
float(sm_VATI_LatLon_DF_row['Latitude']))
        pt_meter = xform.transform(pt_gps)
        # print (pt_meter)
        sm = rlayer.dataProvider().identify(pt_meter, QgsRaster.IdentifyFormatValue)
        sm_val = sm.results()[1]
        # print (sm_val)
        sm_VATI_nparray = np.append(sm_VATI_nparray, sm_val)

    sm_VATI_LatLon_DF['sm_VATI'] = sm_VATI_nparray
    # print (sm_VATI_LatLon_DF.head())

    for index, sm_VATI_LatLon_DF_row in sm_VATI_LatLon_DF.iterrows():
        Plot_Id_val = sm_VATI_LatLon_DF_row['Plot_Id']
        SM_ML_df.loc[(SM_ML_df['Plot_Id'] == Plot_Id_val) & (SM_ML_df['Date'] == date_dfval),
'sm_VATI'] = sm_VATI_LatLon_DF_row['sm_VATI']

writer = ExcelWriter(out_file)
SM_ML_df.to_excel(writer,'SM_ML_values')
writer.save()

print ("done totally")
```

Maximum relative soil moisture:

As saturation capacity is a property of soil at a site, maximum relative soil moisture can be defined as one property of site where the value of relative soil moisture in any time does not exceed the maximum relative soil moisture. Code Snippet used is as follows:

Code Snippet

```
import numpy as np
import pandas as pd
import os
from datetime import datetime, timedelta
from osgeo import gdal, osr
from qgis.core import QgsRasterLayer
from pandas import ExcelWriter

basepath = 'D:\MASTERS-PROJECT\\'
subBasepath = basepath + 'MTECH-PROJ-Prob_1_VATI_vs_Field_Moisture\\'
in_file = subBasepath + 'Soil_Moisture_Berambadi_1617_CropAge_VATI_out.xlsx'
out_file = subBasepath + 'Soil_Moisture_Berambadi_1617_CropAge_VATI_withMinMaxRSM_out.xlsx'

sm_folder = 'D:\MASTERS-PROJECT\sm_VATI\\'
```

```
SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')

RSM_minmax_DF = SM_ML_df[['Plot_Id', 'Latitude', 'Longitude']]

RSM_minmax_DF.drop_duplicates(subset=['Plot_Id'], keep='first', inplace=True)
RSM_minmax_DF.reset_index(drop=True, inplace=True)

# print (RSM_max_DF.shape[0])

rsm_Max_npArray = np.zeros(RSM_minmax_DF.shape[0])
rsm_Min_npArray = np.zeros(RSM_minmax_DF.shape[0])

folder = os.listdir(sm_folder)
fldr_len = len(folder)
for ij, each_file in enumerate(folder):
    if (ij % 30 ==0):
        print (ij, "completed")
    if '.tif' in each_file and '.xml' not in each_file:
        file_fullPath = sm_folder + each_file
        # print (file_fullPath)
        for index, row in RSM_minmax_DF.iterrows():
            # print (index)
            rlayer_rsm = QgsRasterLayer(file_fullPath)
            rsm_ds = gdal.Open(file_fullPath)

            prj = rsm_ds.GetProjectionRef()
            srs = osr.SpatialReference(wkt=prj)
            EPSG_num = srs.GetAttrValue('authority', 1)

            crsSrc = QgsCoordinateReferenceSystem(4326)      # WGS 84
            crsDest = QgsCoordinateReferenceSystem(int(EPSG_num))   # WGS 84 / UTM zone 43N
            xform = QgsCoordinateTransform(crsSrc, crsDest)

            pt_gps = QgsPoint(float(row['Longitude']), float(row['Latitude']))
            pt_meter = xform.transform(pt_gps)

            RSM = rlayer_rsm.dataProvider().identify(pt_meter, QgsRaster.IdentifyFormatValue)
            if (RSM.isValid()):
                if not RSM.results()[1] == None:
                    rsm_val = RSM.results()[1]
                    if(rsm_Max_npArray[index] < rsm_val):
                        rsm_Max_npArray[index] = rsm_val
                    if(rsm_Min_npArray[index] > rsm_val):
                        rsm_Min_npArray[index] = rsm_val

RSM_minmax_DF['min_RSM'] = rsm_Min_npArray
RSM_minmax_DF['max_RSM'] = rsm_Max_npArray
```

```
SM_ML_df['min_RSM'] =
SM_ML_df['Plot_Id'].map(RSM_minmax_DF.set_index('Plot_Id')['min_RSM'])
SM_ML_df['max_RSM'] =
SM_ML_df['Plot_Id'].map(RSM_minmax_DF.set_index('Plot_Id')['max_RSM'])

print ("writing to excel")
writer = ExcelWriter(out_file)
SM_ML_df.to_excel(writer,'SM_ML_values')
writer.save()
print ("done totally")
```

Soil Class:

The soil classes are available as vector file, ie., dbase file. Hence, the easiest approach that is followed is to get the soil class of all sites based on nearest neighbor algorithm. The major classes of the soil has been defined in terms of locations and the description and soil type is mentioned according. To feed into machine learning algorithms, The soil types are mapped to integer types accordingly.

| Soil_Type | Int_Type |
|---|---|
| **Clay** | 1 |
| **Gravelly_Sandy_Clay** | 2 |
| **Gravelly_Sandy_Clay_Loam** | 3 |
| **Sandy_Clay** | 4 |
| **Sandy_Clay_Loam** | 5 |

Table showing soil types and corresponding mapped integer types for ML algorithms

| | Soil Name | Description | Soil_Type | Int_Type |
|---|---|---|---|---|
| **ARK** | Annurkeri soils | well drained, have dark reddish brown to very dusky red sandy clay to clay soils | Sandy_Clay | 4 |
| **BMB** | Beemanabeedu soils | moderately well drained, have very dark greyish brown to dark grey and very dark brown clayey soils | Clay | 1 |
| **BMD** | Berambadi soils | well drained, dark brown to dark greyish brown clayey soils | Clay | 1 |
| **BRG** | Bargi soils | well drained, have very dark brown to very dark grayish brown clay soils | Clay | 1 |
| **DRH** | Devarahalli soils | well drained, have dark red to reddish brown and dusky red gravelly sandy clay loam to sandy clay soils | Gravelly_Sandy_Clay_Loam | 3 |
| **GPR** | Gopalap | well drained, have dark brown to dark reddish brown | Gravelly_S | 3 |

| | | | | |
|---|---|---|---|---|
| | ura soils | and reddish brown gravelly sandy clay loam to sandy clay soils | andy_Clay _Loam | |
| **HDR** | Hindupur soils | well drained, have dark reddish brown to dusky red sandy clay loam to sandy clay soils | Sandy_Clay_Loam | 5 |
| **HGH** | Honnegaudanahalli soils | well drained, have very dark brown to brown and dark reddish brown sandy clay loam soils | Sandy_Clay_Loam | 5 |
| **HPR** | Hullipura soils | well drained, have dark brown to very dark brown gravelly sandy clay loam to sandy clay soils | Gravelly_Sandy_Clay_Loam | 3 |
| **KDH** | Kalligaudanahalli soils | well drained, have dark red to dark reddish brown and dark brown sandy clay to clay soils | Sandy_Clay | 4 |
| **KLP** | Kallipura soils | well drained, have dark reddish brown to dark red gravelly sandy clay loam to sandy clay soils | Gravelly_Sandy_Clay_Loam | 3 |
| **KNG** | Kannigala soils | well drained, have dark reddish brown to dark red gravelly sandy clay loam to sandy clay soils | Gravelly_Sandy_Clay_Loam | 3 |
| **MDH** | Maddinahundi soils | well drained, have dark reddish brown gravelly sandy clay soils | Gravelly_Sandy_Clay | 2 |
| **MGH** | Magoonahalli soils | well drained, have very dark brown to dark brown gravelly sandy clay loam soils | Gravelly_Sandy_Clay_Loam | 3 |
| **SPR** | Shivapura soils | well drained, have dark reddish brown gravelly sandy clay loam to sandy clay soils | Gravelly_Sandy_Clay_Loam | 3 |

Table showing classes and corresponding classification and description

The code snippet used for extracting soil class is as follows:

Code Snippet

```
import numpy as np
import pandas as pd
from pandas import ExcelWriter

basepath = 'D:\MASTERS-PROJECT\\'
subBasepath = basepath + 'MTECH-PROJ-Prob_1_VATI_vs_Field_Moisture\\'
in_file = subBasepath + 'Soil_Moisture_Berambadi_1617_CropAge_VATI_withMinMaxRSM_out.xlsx'
out_file = subBasepath +
'Soil_Moisture_Berambadi_1617_CropAge_VATI_withMinMaxRSM_SoilClass_out.xlsx'

soil_class_file = subBasepath + 'Soil_Classes.xlsx'

soil_class_df = pd.read_excel(soil_class_file, sheet_name='soil_classes')
soil_class_df = soil_class_df[['longitude','latitude','Int_Type']]
# print (soil_class_df)

SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')
```

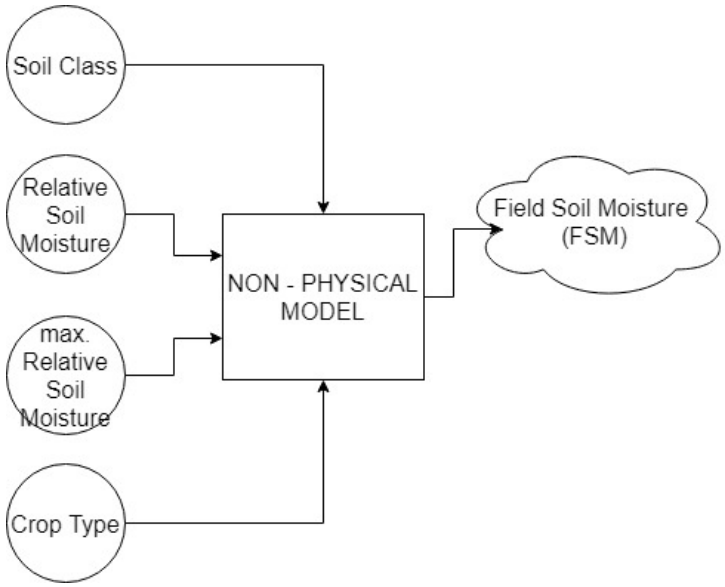
```
sm_Soil_Class_LatLon_DF = SM_ML_df[['Plot_Id', 'Latitude', 'Longitude']]
sm_Soil_Class_LatLon_DF = sm_Soil_Class_LatLon_DF.drop_duplicates(subset=['Plot_Id'], keep='first')
sm_Soil_Class_LatLon_DF.reset_index(drop=True, inplace=True)
# print (sm_Soil_Class_LatLon_DF.shape)

soil_found_class_nparray = np.empty(shape=sm_Soil_Class_LatLon_DF.shape[0])
soil_found_class_nparray[:] = np.nan
# print (soil_found_class_nparray.shape)

count = 1
for i, SM_row in sm_Soil_Class_LatLon_DF.iterrows():
    sm_longitude = SM_row['Longitude']
    sm_latitude = SM_row['Latitude']

    left_extent_arry = np.array([])
    right_extent_arry = np.array([])
    for j, Class_row in soil_class_df.iterrows():
        if sm_longitude >= Class_row['longitude']:
            left_extent_arry = np.append(left_extent_arry, j)
        if sm_longitude <= Class_row['longitude']:
            right_extent_arry = np.append(right_extent_arry, j)

    if (left_extent_arry.shape[0] != 0):
        left_extent_j = left_extent_arry.max()
    else:
        left_extent_j = None
    if (right_extent_arry.shape[0] != 0):
        right_extent_j = right_extent_arry.min()
    else:
        right_extent_j = None

    # print (sm_longitude, left_extent_j, right_extent_j)
    if (left_extent_j != None and right_extent_j != None):
        top_extent_arry = np.array([])
        bottom_extent_arry = np.array([])

        for jk, Class_row in soil_class_df.iterrows():
            if (right_extent_j >= jk >= left_extent_j):
                if sm_latitude >= Class_row['latitude']:
                    bottom_extent_arry = np.append(bottom_extent_arry, jk)
                if sm_latitude <= Class_row['latitude']:
                    top_extent_arry = np.append(top_extent_arry, jk)

        if (top_extent_arry.shape[0] != 0):
            LT_extent_jk = top_extent_arry.min()
        else:
            LT_extent_jk = None
        if (bottom_extent_arry.shape[0] != 0):
            RT_extent_jk = bottom_extent_arry.max()
```

```
        else:
            RT_extent_jk = None

        # taking left top value for soil
        if (LT_extent_jk != None and RT_extent_jk != None):
            soil_found_class_nparray[i] = int(soil_class_df['Int_Type'].iloc[int(LT_extent_jk)])
    else:
        soil_found_class_nparray[i] = np.nan

sm_Soil_Class_LatLon_DF['Int_Type'] = soil_found_class_nparray

SM_ML_df['Soil_Class'] =
SM_ML_df['Plot_Id'].map(sm_Soil_Class_LatLon_DF.set_index('Plot_Id')['Int_Type'])

print ("writing to excel")
writer = ExcelWriter(out_file)
SM_ML_df.to_excel(writer,'SM_ML_values')
```

Thus, the different features that can be used in machine learning are (i) crop type, (ii) maximum relative soil moisture and (iii) soil class along with relative soil moisture obtained from satellite data.

## Methodology

It is general practice to use physical model. In this study, a non physical model is used to analyze this problem. The dataset to be fed to this model consists of around 5200 data points.

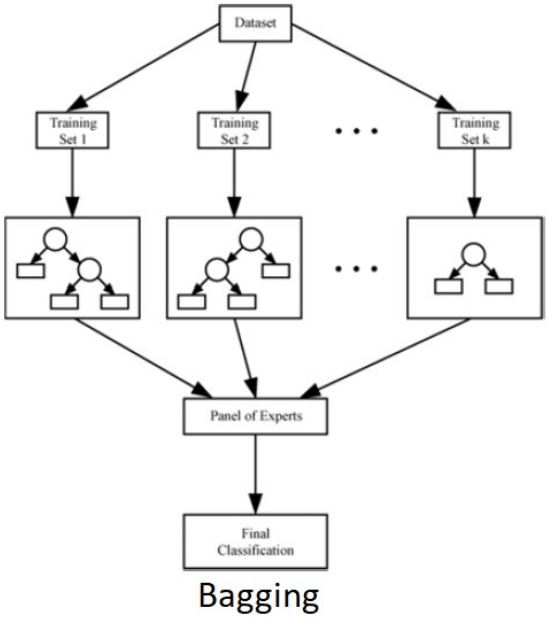



Non-Physical Model

This model has four inputs which are soil class, relative soil moisture, maximum relative soil moisture and crop type. And the output of the model is Field Soil Moisture. To train this model, bagging regressor, artificial neural networks and tensorflow custom machine learning regressor are used and discussed.

**Bagging Regressor:**

BAGGING is coined from 'Bootstrap AGGregatING'. Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator by introducing randomization into its construction procedure and then making an ensemble out of it.



Bagging

Given a set D of d tuples, at each iteration i, a training set $D_i$ of d tuples is sampled with replacement from D. A classifier model $M_i$ is learned for each training set $D_i$. Regression can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple. The code snippet used for bagging regressor is as follows:

```
import pickle
import numpy as np
import pandas as pd
from pandas import ExcelWriter
```

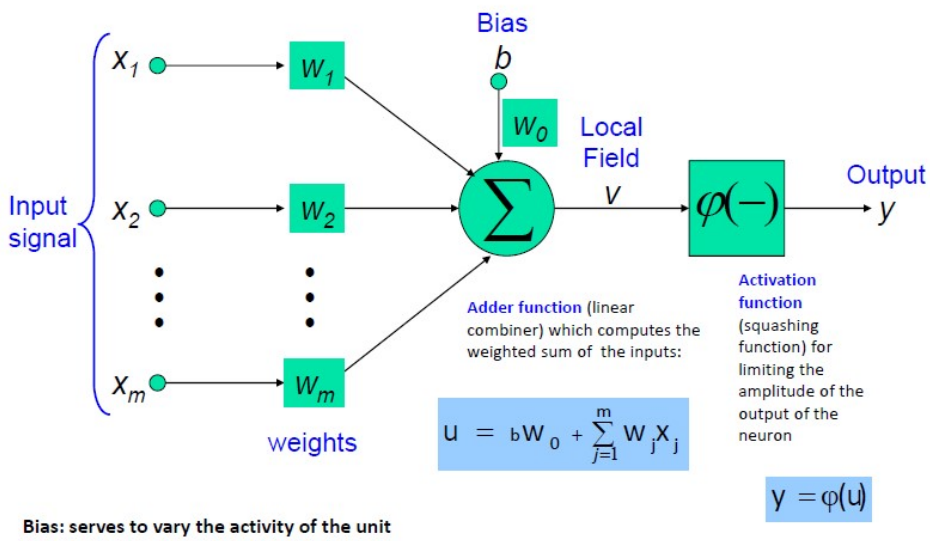
```
import matplotlib.pyplot as plt
import itertools
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor

def load_data(in_file):
    SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')
    reqd_data_arr = ['Plot_Id', 'Crop_Type', 'Field_SM', 'rsm_VATI', 'max_RSM', 'Soil_Class']
    SM_ML_df = SM_ML_df[reqd_data_arr]
    SM_ML_df.replace(0, np.nan, inplace=True)
    SM_ML_df.replace(-np.inf, np.nan, inplace=True)
    SM_ML_df = SM_ML_df.dropna(axis=0, how='any')
    return SM_ML_df

def replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df):
    # Replacing with crop classes
    crop_class_file = basepath + 'd_crop_classes.xlsx'
    crop_class_df = pd.read_excel(crop_class_file, sheet_name='crop_classes')
    # print (crop_class_df)

    replacing_dict = {}
    for index, each_row in crop_class_df.iterrows():
    #     print (each_row, type(each_row))
        replacing_dict[each_row['Crop_Type']] = 'class' + str(each_row['Crop_Class'])
    # print (replacing_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().replace(replacing_dict)
    # print (SM_ML_df)

    # Forming Dictionary for crop Type and replacing with integer in Crop Type Column
    Crop_Type_nparray = SM_ML_df['Crop_Type'].str.strip().unique()
    # print (Crop_Type_nparray)

    int_toCrop_type_dict = dict(enumerate(Crop_Type_nparray))
    crop_type_toInt_dict = {y:x for x,y in int_toCrop_type_dict.items()}
    # print (int_toCrop_type_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().map(crop_type_toInt_dict)
    # print (SM_ML_df)
    return SM_ML_df

def make_data_ready(SM_ML_df):
    X = SM_ML_df[['Crop_Type', 'rsm_VATI', 'Soil_Class']]
    y = SM_ML_df[['Field_SM']]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    X_train, X_test, y_train, y_test = X_train.values, X_test.values, y_train.values.flatten(),
y_test.values.flatten()
    return (X_train, X_test, y_train, y_test)

def bagging(X_train, X_test, y_train, y_test):
```

```
    clf = BaggingRegressor()
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    return (clf, score)

def plot_y_vs_y_predicted(clf, X_test, y_test):
    y_predict = clf.predict(X_test)
    v = [0, 40, 0, 40]
    plt.axis(v)
    plt.scatter(y_test, y_predict)
    plt.xlabel("Y_Actual")
    plt.ylabel("Y_Predicted")
    plt.show()

def main_func():
#     basepath = 'C:\\Users\\soi\\Documents\\MTECH-PROJ-Phase-2\\'
#     basepath = 'C:\\Users\\rukmangadan\\Documents\\MTECH-PROJ-Phase-2\\'
    basepath = 'C:\\Users\\theorist\\Documents\\MTECH-PROJ-Phase-2\\'
    in_file = basepath + 'd_soil_moisture_comparison_data.xlsx'
    SM_ML_df = load_data(in_file)
    SM_ML_df = replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df)
    X_train, X_test, y_train, y_test = make_data_ready(SM_ML_df)
    clf, score = bagging(X_train, X_test, y_train, y_test)
    print(score)
    plot_y_vs_y_predicted(clf, X_test, y_test)
    plot_y_vs_y_predicted(clf, X_train, y_train)
```

## Artificial Neural Networks:

A neural network is a set of connected input/output units (neurons) where each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights that enable it to predict the correct class label of the input samples.

Neural networks are built out of a densely interconnected set of simple units (neurons).




- Each neuron takes a number of real-valued inputs.
- Produces a single real-valued output.
- Inputs to a neuron may be the outputs of other neurons.
- A neuron's output may be used as input to many other neurons.

Artificial Neural Networks can also be used to learn complex pseudo random number generator. The code snippet used for artificial neural networks is as follows:

```
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

def load_data(in_file):
    SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')
    reqd_data_arr = ['Plot_Id', 'Crop_Type', 'Field_SM', 'rsm_VATI', 'max_RSM', 'Soil_Class']
    SM_ML_df = SM_ML_df[reqd_data_arr]
    SM_ML_df.replace(0, np.nan, inplace=True)
    SM_ML_df.replace(-np.inf, np.nan, inplace=True)
    SM_ML_df = SM_ML_df.dropna(axis=0, how='any')
    return SM_ML_df

def replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df):
    # Replacing with crop classes
    crop_class_file = basepath + 'd_crop_classes.xlsx'
    crop_class_df = pd.read_excel(crop_class_file, sheet_name='crop_classes')
    # print (crop_class_df)
```

```
    replacing_dict = {}
    for index, each_row in crop_class_df.iterrows():
    #     print (each_row, type(each_row))
        replacing_dict[each_row['Crop_Type']] = 'class' + str(each_row['Crop_Class'])
#     print (replacing_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().replace(replacing_dict)
    # print (SM_ML_df)

    # Forming Dictionary for crop Type and replacing with integer in Crop Type Column
    Crop_Type_nparray = np.sort(SM_ML_df['Crop_Type'].str.strip().unique())
#     print (Crop_Type_nparray)

    int_toCrop_type_dict = dict(enumerate(Crop_Type_nparray))
    crop_type_toInt_dict = {y:(x+1) for x,y in int_toCrop_type_dict.items()}
#     print (int_toCrop_type_dict)
#     print(crop_type_toInt_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().map(crop_type_toInt_dict)
#     print (SM_ML_df)
#     print(SM_ML_df['Crop_Type'].unique())
#     print(SM_ML_df['Soil_Class'].unique())
    return SM_ML_df

def make_data_ready(SM_ML_df):
    X = SM_ML_df[['Crop_Type', 'rsm_VATI', 'max_RSM', 'Soil_Class']]
    y = SM_ML_df[['Field_SM']]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    X_train, X_test, y_train, y_test = X_train.values, X_test.values, y_train, y_test
    return (X_train, X_test, y_train, y_test)

# basepath = 'C:\\Users\\soi\\Documents\\MTECH-PROJ-Phase-2\\'
basepath = 'C:\\Users\\rukmangadan\\Documents\\MTECH-PROJ-Phase-2\\'
# basepath = 'C:\\Users\\theorist\\Documents\\MTECH-PROJ-Phase-2\\'
in_file = basepath + 'd_soil_moisture_comparison_data.xlsx'
SM_ML_df = load_data(in_file)
SM_ML_df = replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df)
#     print(SM_ML_df)
X_train, X_test, y_train, y_test = make_data_ready(SM_ML_df)

def tnsrflw(train_input, y_train, test_input, optimizer_type, epochs):
    train_output = y_train.values.flatten()

    A = tf.placeholder("float", [None, 4])
    Y = tf.placeholder("float", [None, 1])

    layer_1 = tf.layers.dense(A, 256)
    layer_2 = tf.layers.dense(layer_1, 256)
    layer_3 = tf.layers.dense(layer_2, 256)
```

```
    layer_4 = tf.layers.dense(layer_3, 256)
    layer_5 = tf.layers.dense(layer_4, 256)
    pred = tf.layers.dense(layer_5, 1)

    cost = tf.reduce_max(tf.square(pred - Y))
    optimizer = optimizer_type.minimize(cost)

    init = tf.global_variables_initializer()

    sess = tf.Session()
    sess.run(init)

    step = 0
    total_steps = train_input.shape[0] * epochs
    for epoch in range(epochs):
        for (x, y) in zip(train_input, train_output):
            sess.run(optimizer, feed_dict={A:[[x[0],x[1],x[2],x[3]]], Y:[[y]]})
            if(step % (total_steps/4) == 0):
                print("Step", step, "of", total_steps, ":")
                print ("Cost: ",sess.run(cost,
feed_dict={A:np.transpose([train_input[:,0],train_input[:,1],train_input[:,2],train_input[:,3]]),
Y:np.transpose([train_output])}),"\n")
            step+=1

    print("Step", total_steps, "of", total_steps, ":")
    print ("Cost: ",sess.run(cost,
feed_dict={A:np.transpose([train_input[:,0],train_input[:,1],train_input[:,2],train_input[:,3]]),
Y:np.transpose([train_output])}),"\n")

    y_pred_test = sess.run(pred,
feed_dict={A:np.transpose([test_input[:,0],test_input[:,1],test_input[:,2],test_input[:,3]])})
    y_pred_train = sess.run(pred,
feed_dict={A:np.transpose([train_input[:,0],train_input[:,1],train_input[:,2],train_input[:,3]])})

    sess.close()
    return y_pred_test, y_pred_train

optimizer_type = tf.train.AdamOptimizer(0.001)
epochs = 1000
y_pred_test, y_pred_train = tnsrflw(X_train, y_train, X_test, optimizer_type, epochs)

def plot_y_vs_y_predicted(y_actual, y_pred):
    y_actual_val = y_actual.values.flatten()
    v = [0, 40, 0, 40]
    plt.axis(v)
    plt.scatter(y_actual_val, y_pred)
    plt.xlabel("Y_Actual")
    plt.ylabel("Y_Predicted")
    plt.show()
```

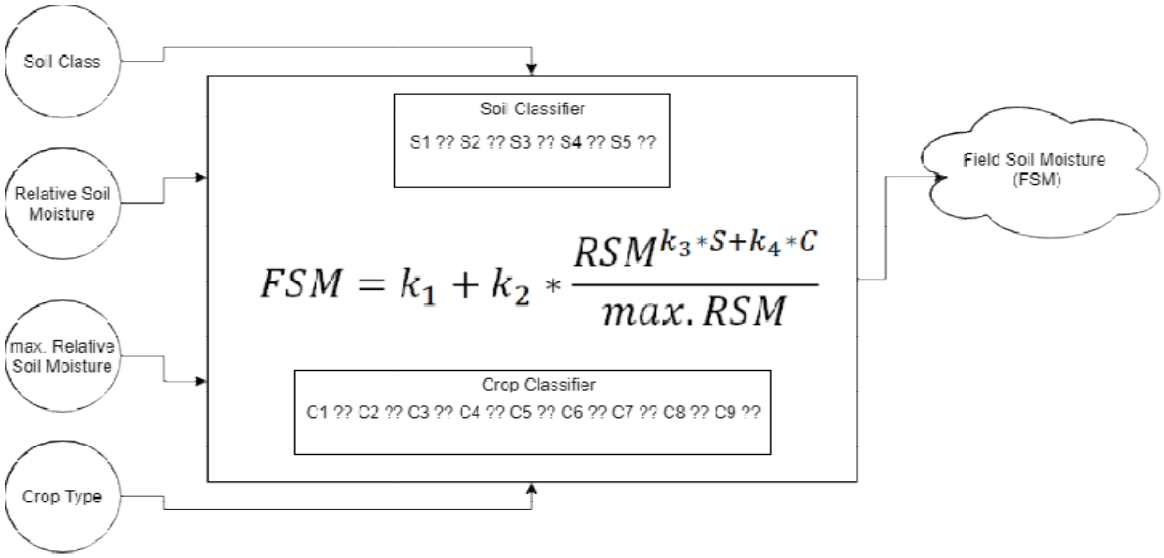
```
def number_of_points_statistics(y_test, y_pred_test, y_train, y_pred_train):
    y_test_val = y_test.values.flatten()
    y_train_val = y_train.values.flatten()
    no_of_points_above = 0
    no_of_points_matching = 0
    no_of_points_below = 0

    for Y1, Y2 in zip(y_test_val, y_pred_test):
        if(0.7 * Y2 < Y1 and Y1 < 1.3 * Y2):
            no_of_points_matching += 1
        elif (Y1 < Y2):
            no_of_points_above += 1
        elif (Y2 < Y1):
            no_of_points_below += 1

    for Y1, Y2 in zip(y_train_val, y_pred_train):
        if(0.7 * Y2 < Y1 and Y1 < 1.3 * Y2):
            no_of_points_matching += 1
        elif (Y1 < Y2):
            no_of_points_above += 1
        elif (Y2 < Y1):
            no_of_points_below += 1

    print("no_of_points_matching: ", no_of_points_matching)
    print("no_of_points_above: ", no_of_points_above)
    print("no_of_points_below: ", no_of_points_below)

number_of_points_statistics(y_test, y_pred_test, y_train, y_pred_train)
print("Test Data Graph:")
plot_y_vs_y_predicted(y_test, y_pred_test)
print("Trained Data Graph:")
plot_y_vs_y_predicted(y_train, y_pred_train)
```

## Tensorflow Machine Learning Custom Regressor:

Tensorflow has a custom regressor module wherein one can input the regressor functions along with variables and the corresponding training dataset for the model to train itself. Also, it has option of controlling flow of data for training based on conditions. The cost function for this model will be simple RMSE (Root Mean Square Error) between the predicted and actual values. The optimizer used for reducing RMSD (Root Mean Square Deviation) is Gradient Descent Optimizer.

Regression ML Model:

Since the dataset comprises of five soil classes and nine crop classes, 13 tensorflow variables will be kept for each class of S & C, ie., S1, S2, S3, S4, S5 for soil classes and C1, C2, …,C9 for crop classes.




Machine Learning Model

The Field Soil Moisture (FSM) can be computed using

$$FSM = k_1 + k_2 * \frac{RSM^{k_3*S+k_4*C}}{max.RSM}$$

where

$k_1$, $k_2$, $k_3$, $k_4$ → constants of regression model

S → coefficient corresponding to type of soil

C → coefficient corresponding to type of crop

RSM → Relative Soil Moisture obtained from the satellite data

max.RSM → maximum RSM for the field point

The code snippet used for Tensorflow ML regressor is as follows:

```
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

def load_data(in_file):
    SM_ML_df = pd.read_excel(in_file, sheet_name='SM_ML_values')
    reqd_data_arr = ['Plot_Id', 'Crop_Type', 'Field_SM', 'rsm_VATI', 'max_RSM', 'Soil_Class']
    SM_ML_df = SM_ML_df[reqd_data_arr]
    SM_ML_df.replace(0, np.nan, inplace=True)
    SM_ML_df.replace(-np.inf, np.nan, inplace=True)
    SM_ML_df = SM_ML_df.dropna(axis=0, how='any')
    return SM_ML_df

def replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df):
    # Replacing with crop classes
    crop_class_file = basepath + 'd_crop_classes.xlsx'
    crop_class_df = pd.read_excel(crop_class_file, sheet_name='crop_classes')
    # print (crop_class_df)

    replacing_dict = {}
    for index, each_row in crop_class_df.iterrows():
    #     print (each_row, type(each_row))
        replacing_dict[each_row['Crop_Type']] = 'class' + str(each_row['Crop_Class'])
#     print (replacing_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().replace(replacing_dict)
    # print (SM_ML_df)

    # Forming Dictionary for crop Type and replacing with integer in Crop Type Column
    Crop_Type_nparray = np.sort(SM_ML_df['Crop_Type'].str.strip().unique())
#     print (Crop_Type_nparray)

    int_toCrop_type_dict = dict(enumerate(Crop_Type_nparray))
    crop_type_toInt_dict = {y:(x+1) for x,y in int_toCrop_type_dict.items()}
#     print (int_toCrop_type_dict)
#     print(crop_type_toInt_dict)

    SM_ML_df["Crop_Type"] = SM_ML_df["Crop_Type"].str.strip().map(crop_type_toInt_dict)
#     print (SM_ML_df)
#     print(SM_ML_df['Crop_Type'].unique())
#     print(SM_ML_df['Soil_Class'].unique())
    return SM_ML_df

def make_data_ready(SM_ML_df):
    X = SM_ML_df[['Crop_Type', 'rsm_VATI', 'max_RSM', 'Soil_Class']]
    y = SM_ML_df[['Field_SM']]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    X_train, X_test, y_train, y_test = X_train.values, X_test.values, y_train, y_test
```

```
    return (X_train, X_test, y_train, y_test)

def tnsrflw(train_input, y_train, test_input, optimizer_type, epochs):
    train_output = y_train.values.flatten()

    k1 = tf.Variable(tf.random_normal([1]))
    k2 = tf.Variable(tf.random_normal([1]))
    k3 = tf.Variable(tf.random_normal([1]))
    k4 = tf.Variable(tf.random_normal([1]))
    k5 = tf.Variable(tf.random_normal([1]))

    c1 = tf.Variable(tf.random_normal([1]))
    c2 = tf.Variable(tf.random_normal([1]))
    c3 = tf.Variable(tf.random_normal([1]))
    c4 = tf.Variable(tf.random_normal([1]))
    c5 = tf.Variable(tf.random_normal([1]))
    c6 = tf.Variable(tf.random_normal([1]))
    c7 = tf.Variable(tf.random_normal([1]))
    c8 = tf.Variable(tf.random_normal([1]))
    c9 = tf.Variable(tf.random_normal([1]))

    s1 = tf.Variable(tf.random_normal([1]))
    s2 = tf.Variable(tf.random_normal([1]))
    s3 = tf.Variable(tf.random_normal([1]))
    s4 = tf.Variable(tf.random_normal([1]))
    s5 = tf.Variable(tf.random_normal([1]))

    A = tf.placeholder("float")
    B = tf.placeholder("float")
    C = tf.placeholder("float")
    D = tf.placeholder("float")

    Y = tf.placeholder("float")

    s2_s1 = tf.where(tf.logical_and(1.9<D,D<2.1),x=s2,y=s1)
    s3_s2 = tf.where(tf.logical_and(2.9<D,D<3.1),x=s3,y=s2_s1)
    s4_s3 = tf.where(tf.logical_and(3.9<D,D<4.1),x=s4,y=s3_s2)
    s = tf.where(tf.logical_and(4.9<D,D<5.1),x=s5,y=s4_s3)

    c2_c1 = tf.where(tf.logical_and(1.9<A,A<2.1),x=c2,y=c1)
    c3_c2 = tf.where(tf.logical_and(2.9<A,A<3.1),x=c3,y=c2_c1)
    c4_c3 = tf.where(tf.logical_and(3.9<A,A<4.1),x=c4,y=c3_c2)
    c5_c4 = tf.where(tf.logical_and(4.9<A,A<5.1),x=c5,y=c4_c3)
    c6_c5 = tf.where(tf.logical_and(5.9<A,A<6.1),x=c6,y=c5_c4)
    c7_c6 = tf.where(tf.logical_and(6.9<A,A<7.1),x=c7,y=c6_c5)
    c = tf.where(tf.logical_and(7.9<A,A<8.1),x=c8,y=c7_c6)

    pred = k1 + k2 * tf.pow(B, k3 * s + k4 * c) / tf.pow(C, k5)

    cost = tf.reduce_max(tf.square(pred - Y))
```

```
    optimizer = optimizer_type.minimize(cost)

    init = tf.global_variables_initializer()

    sess = tf.Session()
    sess.run(init)

    step = 0
    total_steps = train_input.shape[0] * epochs
    for epoch in range(epochs):
        for (x, y) in zip(train_input, train_output):
#             print(int(x[3]))
            sess.run(optimizer, feed_dict={A:x[0],B:x[1],C:x[2],D:x[3], Y:y})
            if(step % (total_steps/4) == 0):
                print("Step", step, "of", total_steps, ":")
                k = sess.run([k1, k2, k3, k4, k5])
                s = sess.run([s1, s2, s3, s4, s5])
                c = sess.run([c1, c2, c3, c4, c5, c6, c7, c8])
                print (k[0], k[1], k[2], k[3], k[4])
                print (s[0], s[1], s[2], s[3], s[4])
                print (c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7])
                print ("Cost: ",sess.run(cost, feed_dict={A:x[0],B:x[1],C:x[2],D:x[3], Y:y}),"\n")
            step+=1

    print("Step", total_steps, "of", total_steps, ":")
    k = sess.run([k1, k2, k3, k4, k5])
    s = sess.run([s1, s2, s3, s4, s5])
    c = sess.run([c1, c2, c3, c4, c5, c6, c7, c8])
    print (k[0], k[1], k[2], k[3], k[4])
    print (s[0], s[1], s[2], s[3], s[4])
    print (c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7],"\n")

    y_pred_test = np.array([])
    for tst_inpt in test_input:
        y_pred_test = np.append(y_pred_test, sess.run(pred,
feed_dict={A:tst_inpt[0],B:tst_inpt[1],C:tst_inpt[2],D:tst_inpt[3]}))

    y_pred_trn = np.array([])
    for trn_inpt in train_input:
        y_pred_trn = np.append(y_pred_trn, sess.run(pred,
feed_dict={A:trn_inpt[0],B:trn_inpt[1],C:trn_inpt[2],D:trn_inpt[3]}))

    sess.close()
    return y_pred_test, y_pred_trn

def plot_y_vs_y_predicted(y_actual, y_pred):
    y_actual_val = y_actual.values.flatten()
    v = [0, 40, 0, 40]
    plt.axis(v)
    plt.scatter(y_actual_val, y_pred)
```

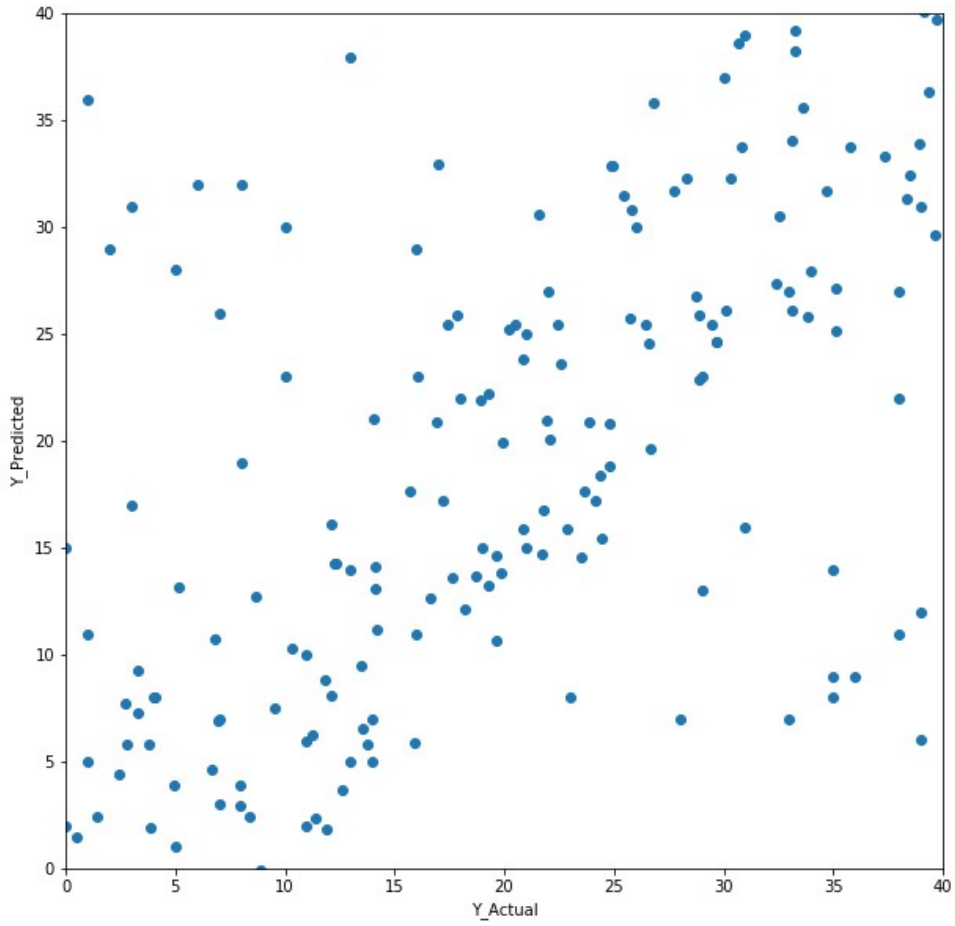
```
    plt.xlabel("Y_Actual")
    plt.ylabel("Y_Predicted")
    plt.show()

def number_of_points_statistics(y_test, y_pred_test, y_train, y_pred_trn):
    y_test_val = y_test.values.flatten()
    y_train_val = y_train.values.flatten()
    no_of_points_above = 0
    no_of_points_matching = 0
    no_of_points_below = 0

    for Y1, Y2 in zip(y_test_val, y_pred_test):
        if(0.7 * Y2 < Y1 and Y1 < 1.3 * Y2):
            no_of_points_matching += 1
        elif (Y1 < Y2):
            no_of_points_above += 1
        elif (Y2 < Y1):
            no_of_points_below += 1

    for Y1, Y2 in zip(y_train_val, y_pred_trn):
        if(0.7 * Y2 < Y1 and Y1 < 1.3 * Y2):
            no_of_points_matching += 1
        elif (Y1 < Y2):
            no_of_points_above += 1
        elif (Y2 < Y1):
            no_of_points_below += 1

    print("no_of_points_matching: ", no_of_points_matching)
    print("no_of_points_above: ", no_of_points_above)
    print("no_of_points_below: ", no_of_points_below)

# basepath = 'C:\\Users\\soi\\Documents\\MTECH-PROJ-Phase-2\\'
basepath = 'C:\\Users\\rukmangadan\\Documents\\MTECH-PROJ-Phase-2\\'
# basepath = 'C:\\Users\\theorist\\Documents\\MTECH-PROJ-Phase-2\\'
in_file = basepath + 'd_soil_moisture_comparison_data.xlsx'
SM_ML_df = load_data(in_file)
SM_ML_df = replace_df_with_crop_classes_andMaptoInt(basepath, SM_ML_df)
#     print(SM_ML_df)

X_train, X_test, y_train, y_test = make_data_ready(SM_ML_df)
optimizer_type = tf.train.AdamOptimizer(0.001)
epochs = 100
y_pred_test, y_pred_trn = tnsrflw(X_train, y_train, X_test, optimizer_type, epochs)
number_of_points_statistics(y_test, y_pred_test, y_train, y_pred_trn)
print("Test Data Graph:")
plot_y_vs_y_predicted(y_test, y_pred_test)
print("Trained Data Graph:")
plot_y_vs_y_predicted(y_train, y_pred_trn)
print(y_test)
```

# Chapter 4

## Results and Discussions

*Bagging Regressor:*

Bagging Regressor gives an average correlation of 0.65. Bagging regressor can also give average results if the dataset has a complex structure. To test if the dataset has a complex structure, Artificial Neural Networks (ANNs) are used.

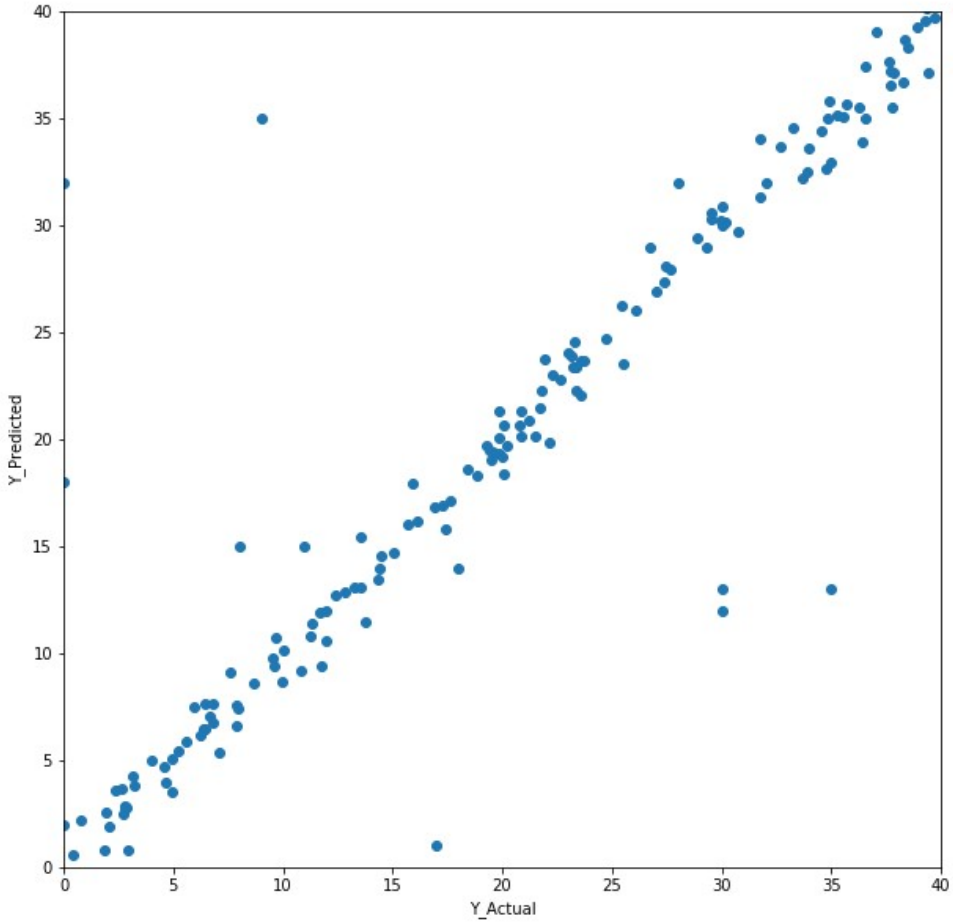



Scatter plot of Bagging Regressor Prediction

- It is far better than a single classifier derived from total dataset.
- For noisy data, it works average only.

*Artificial Neural networks:*

Artificial Neural Networks gives correlation of around 0.92. The results are of high accuracy. This means that a complex structure is found in the dataset.

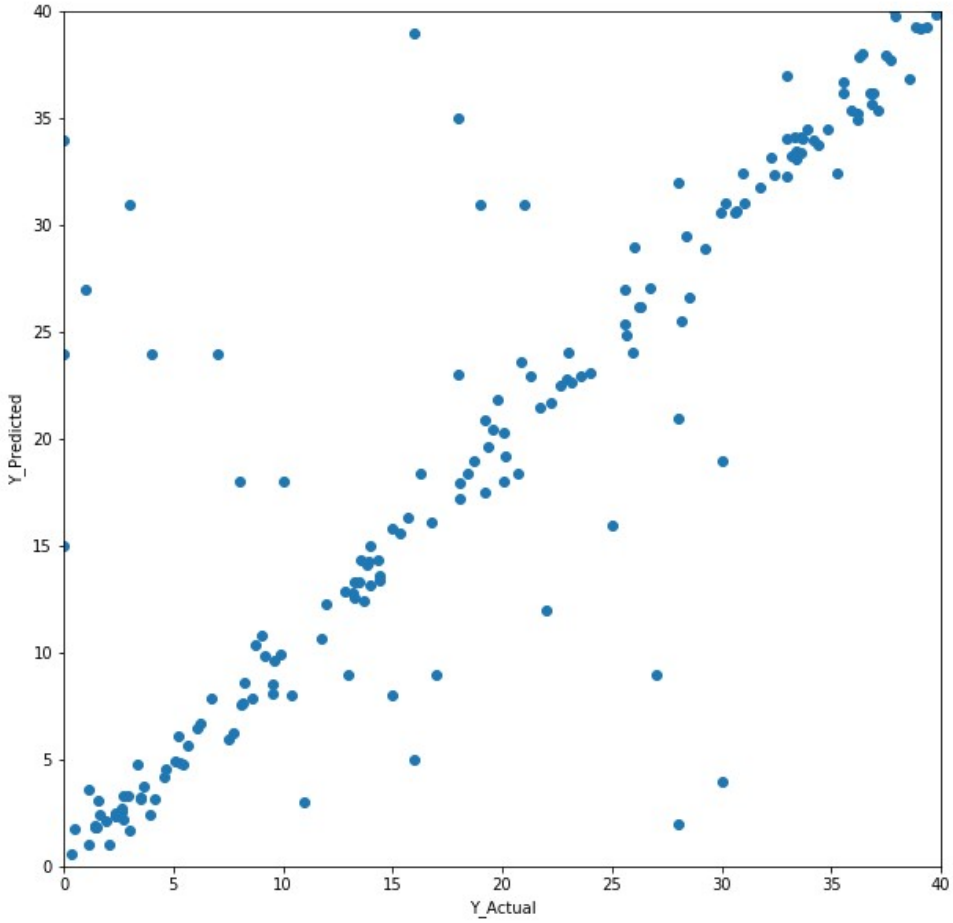



Scatter plot of ANN Prediction

- Prediction accuracy is generally high.
- robust, works when training examples contain errors or noisy data
- parameters are best determined empirically and longer training time
- difficult to understand the learned function (weights)
- not easy to incorporate domain knowledge

*Regression ML Model:*

Artificial Neural Networks has shown that there is a complex structure. Hence, the custom regressor model is tried and regression ML model gives correlation of around 0.86.




For the custom regression model,

$$FSM = k_1 + k_2 * \frac{RSM^{k_3 * S + k_4 * C}}{max.RSM}$$

the values of coefficients are found to be

Constants of regression model:

| Constant | Value |
|---|---|
| k1 | 0.539 |
| k2 | 0.876 |
| k3 | 0.32 |
| k4 | 0.84 |

Coefficients of Soil Class:

| Soil Class | Coefficient | Value |
|---|---|---|
| 1 | s1 | -0.22715 |
| 2 | s2 | -0.27834 |
| 3 | s3 | -0.32364 |
| 4 | s4 | -0.46348 |
| 5 | s5 | -0.39676 |

Coefficients of Crop Class:

| Crop Class | Coefficient | Value |
|---|---|---|
| 1 | c1 | 2.065574 |
| 2 | c2 | 2.328851 |
| 3 | c3 | 2.414203 |
| 4 | c4 | 2.475283 |
| 5 | c5 | 2.439513 |
| 6 | c6 | 2.055184 |
| 7 | c7 | 2.406972 |
| 8 | c8 | 2.342806 |
| 9 | c9 | 2.843768 |

Keeping the tolerance ± 0.5 % of the predicted values, around 20% of the data is showing different values other than the actual values. Banana and Sugarcane which have been classified as tall and more dense is showing higher values. And whichever is showing lesser values, belongs to tall and less dense sunflower and maize plants.

# Chapter 5

## Conclusion

- Thus we are able achieve good correlation between relative soil moisture and field soil moisture to a greater extent which did not make any sense before
- The algorithms could be used for prediction of future values of field moisture content from satellite data.
- Using machine learning, we could extract many interesting features from the datasets also.

## References


(1) H. McNairna,*, C. Duguayb, B. Briscoc, T.J. Pultza (2002). The effect of soil and crop residue characteristics on polarimetric radar response. Remote Sensing of Environment, Volume 80, Issue 2, Pages 308-320.

(2) Jun Wen, Zhongbo Su (2003). A time series based method for estimating relative soil moisture with ERS wind scatterometer data. Geophysical Research Letters, Volume 30, Issue 7.

(3) B.J. Choudury, T.J. Schmugge, R.W. Newton and A.Chang (1978). Effect of Surface Roughness on the Microwave Emission from Soils. Journal of Geophysical Research, Volume 84, Issue C9, Pages 5699–5706

(4) J.R. Wang, P.E. O'Neill, T.J. Jackson, E.T. Engman (1982). A Multi-Frequency Measurement of Thermal Microwave Emission From Soils: The Effects of Soil Texture and Surface Roughness. International Geoscience and Remote Sensing Symposium, Munich.

(5) L. Breiman, “Pasting small votes for classification in large databases and on-line”, Machine Learning, 36(1), 85-103, 1999.

(6) L. Breiman, “Bagging predictors”, Machine Learning, 24(2), 123-140, 1996.

(7) T. Ho, “The random subspace method for constructing decision forests”, Pattern Analysis and Machine Intelligence, 20(8), 832-844, 1998.

(8) G. Louppe and P. Geurts, “Ensembles on Random Patches”, Machine Learning and Knowledge Discovery in Databases, 346-361, 2012.

(9) P. Geurts, D. Ernst., and L. Wehenkel, “Extremely randomized trees”, Machine Learning, 63(1), 3-42, 2006.

(10) J. Friedman, Greedy Function Approximation: A Gradient Boosting Machine, The Annals of Statistics, Vol. 29, No. 5, 2001.

(11) Friedman, Stochastic Gradient Boosting, 1999

(12) T. Hastie, R. Tibshirani and J. Friedman. Elements of Statistical Learning Ed. 2, Springer, 2009.