TÉCNICO LISBOA

Contributions for Soil Nutrient Estimation Using Earth Observation Data: Datasets, Toolkits and Semi-Supervised Learning

Rafael António Fernandes e Costa Marques Candeias

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Dr. Amâncio Lucas de Sousa Pereira

Examination Committee

Chairperson: Prof. Rui Filipe Fernandes Prada
Supervisor: Dr. Amâncio Lucas de Sousa Pereira
Members of the Committee: Prof. Augusto Emanuel Abreu Esteves
Prof. Filipe Magno Gouveia Quintal

May 2024

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Quero agradecer a todos os meus familiares e amigos que me deram forças e motivação para realizar esta dissertação. Um especial agradecimento ao meu mentor, Dr. Amâncio Lucas de Sousa Pereira, que se mostrou sempre disponível e com um espírito crítico que impulsionou a investigação.

Em memória da minha avó, Maria Luísa Jesus Marques Candeias, que faleceu com 83 anos durante a escrita desta tese. Que a sua paz seja eterna.

Abstract

This thesis aims to contribute to the Artificial Intelligence (AI) community by investigating how Machine Learning (ML) predicts soil fertility through Earth Observation Satellites (EOS) data. This involves discovering a dataset with soil fertility, improving a python toolkit, and comparing Supervised Learning (SL) with their Semi-Supervised Learning (SSL) adaptations. We began by understanding soil fertility, which correlates to the concentration of Phosphorus (P), Nitrogen (N), and Potassium (K). Our first objective began by evaluating public datasets. They had to be reliable, with reasonable data and P, N, and K concentrations. Four potential datasets were found: Africa Soil information Service (AfSis), Big Earth Net (BEN), National Cooperative Soil Survey (NCSS), and Land Use and Coverage Area frame Survey (LUCAS). AfSis lacked reliability, BEN was dispersed without land cover classification, and NCSS had insufficient data. Contrarily, LUCAS was reliable, provided land cover information, and had P concentrations. Regarding the python toolkit, TerraSenseTK (tstk) went through attribute renaming, restructuring, new methods, SSL incorporation, and optimization. After being improved, tstk was used to compare SL models with their SSL versions on P prediction with the created dataset. Several experiments provided a variety of results and interpretations. Our research concludes with a high-quality dataset containing $16,033$ EOS data points spread across Europe, each with P concentrations. The tstk modifications make it user-friendly, practical, and facilitate EOS extraction and subsequent ML procedures. Our study demonstrated that SSL models slightly outperformed traditional SL models in estimating P values. Additionally, performance improved on balanced data of the same land cover ($Triticum\ spelta$). Further research is needed to validate our findings.

Keywords

Semi-Supervised Learning, Nutrients, Earth-Observation-Satellite, Python-Toolkit, Datasets, Spectral-

Vegetation-Indices

Resumo

Esta tese serve de contribuição para a comunidade de Artificial Intelligence (AI) que investiga modelos de Machine Learning (ML) que prevêm a fertilidade do solo atravéz de dados Earth Observation Satellites (EOS). Tal propósito requer a aquisição de um dataset útil, melhoria de um toolkit, e comparação de modelos Supervised Learning (SL) com Semi-Supervised Learning (SSL). Para alcançar o primeiro objectivo, avaliámos um conjunto de datasets públicos. Quatro destacaram-se durante esta procura: Africa Soil information Service (AfSis), Big Earth Net (BEN), National Cooperative Soil Survey (NCSS), e Land Use and Coverage Area frame Survey (LUCAS). O AfSis suscitou pouca confiança, o BEN dispersa e não especifica a cobertura dos solos, e o NCSS possui poucos dados. Pelo contrário, o LUCAS transmite segurança, cobertura de solos e ainda concentrações de Phosphorus (P). No que toca o toolkit de python, o TerraSenseTK (tstk) sofreu uma renomeação de atributos, reestruturação, novos métodos, incorporação de SSL e optimizações. Após a sua melhoria, o toolkit foi utilizado para comparar o desempenho dos modelos SSL com as suas versões de SL. Diversas experiências foram realizadas de modo a guarantir um leque de resultados e interpretações. Terminamos a dissertação concluíndo com um dataset de alta qualidade, constituído por 16.033 registos espalhados pela Europa e com concentrações de P. O toolkit tornou-se mais user-friendly, prático, e de fácil utilização para extração de EOS e da sua subsequente aplicação em ML. O nosso estudo demonstra que SSL supera ligeiramente os seus modelos de SL. Também se retira que os resultados melhoram quando os modelos treinam numa só cobertura de solo ($Triticum\ spelta$). Contudo, o estudo requer mais validação.

Palavras Chave

Semi-Supervised Learning, Nutrientes, Observação-Espacial-Satélites, Python-Toolkit, Datasets, Índices Espectrais de Vegetação

Contents

List of Figures

List of Tables

Acronyms

CEC	Cation Exchange Capacity
AEC	Anion Exchange Capacity
SVI	Spectral Vegetation Indices
R	Red
B	Blue
G	Green
NIR	Near-Infra-Red
NDVI	Normalized Difference Vegetation Index
FAO	Food and Agriculture Organization
LUCAS	Land Use and Coverage Area frame Survey
ESDAC	European Soil Data Centre
ESA	European Space Agency
EU	European Union
NASA	National Aeronautics and Space Administration
JAXA	Japan Aerospatial Exploration Agency
ISRO	Indian Space Research Organisation
CNSA	Chinese National Space Administration
OWID	Our World in Data
NCSS	National Cooperative Soil Survey
IFDC	International Fertilizer Development Center
AfSis	Africa Soil information Service
BEN	Big Earth Net

EthioSis	Ethiopia Soil Information Service
NiSis	Nigeria Soil Information Service
IFDC	International Fertilizer Development Center
OAF	One Acre Fund
UC	University of California
RSiM	Remote Sensing Image Analysis
DIMA	Database Systems and Information Management
TU-B	Technische Universität Berlin
C	Carbon
H	Hydrogen
O	Oxygen
N	Nitrogen
P	Phosphorus
K	Potassium
Ca	Calcium
Mg	Magnesium
S	Sulfur
B	Boron
Cu	Copper
Cl	Chlorine
Fe	Iron
Mn	Manganese
Mo	Molybdenum
Zn	Zinc
CO2	Carbon dioxide
NO3-	Inorganic nitrate
NH4+	Ammonium
ppm	parts per million
EOS	Earth Observation Satellites

ISS	International Space Station
GO	Geostationary Orbit
LEO	Low Earth Observation Orbit
MEO	Medium Earth Observation Orbit
SSO	Sun Synchronous Orbit
S2	Sentinel-2
S3	Sentinel-3
LMSS	Landsat 1-5 MSS
LTM	Landsat 4-5 TM
LETM	Landsat 7 ETM+
LOLI	Landsat 8 OLI
L9	Landsat 9
EMS	Electromagnetic Spectrum
MSI	Multispectral Instruments
RGB	Red-Green-Blue
AI	Artificial Intelligence
ML	Machine Learning
SL	Supervised Learning
SSL	Semi-Supervised Learning
UL	Unsupervised Learning
DL	Deep Learning
PCA	Principal Component Analysis
RMSE	Root Mean Square Error
SVM	Support Vector Machine
LS-SVM	Least-Square Support Vector Machine
RF	Random Forest
ANN	Artificial Neural Network
MLR	Multiple Linear Regression
KNN	K-Nearest Neighbours
TP	True Positive

TN	True Negative
FP	False Positive
FN	False Negative
RMSE	Root Mean Squared Error
CCC	Concordance Correlation Coefficient
DS	Data Science
NB	Naïve Bayes
CNB	Complement Naïve Bayes
RF	Random Forest
AdaBoost	Adaptive Boosting
GradBoost	Gradient Boosting
IQR	Interquartile Range
RFE	Recursive Feature Elimination
sklearn	Scikit-learn
tstk	TerraSenseTK
USB	Unified Semi-Supervised Learning Benchmark
LAMDA	LAMDA-SSL
CSV	Comma-Separated Values

1

Introduction

Contents

In this introductory chapter, the reader may find the core purpose, objective and outline of this research work.

1.1 Context

Before interplanetary exploration, soil was defined as *"unconsolidated mineral or organic material on the immediate surface of the earth that serves as a natural medium for the growth of land plants"* [10]. This definition was later deemed outdated and inconsistent for being exclusive to Earth and ignoring consolidated soils. A revised definition describes soil as *"the layer(s) of generally loose mineral and/or organic material affected by physical, chemical, and/or biological processes at or near the planetary surface, usually holding liquids, gases, and biota, and supporting plants"* [10].

Soil is a crucial pillar to our economy, society, and Earth's environment, as it supplies materials for human usage, provides food for livestock, fuel, fiber for clothing, and retains Carbon (C) emissions. It is so important that, as of today, it represents 4% of global income, accounting to 25% of the least developing countries[1]. Moreover, it is estimated that the human population will grow by a third until 2050 [11]. Due to this unprecedented rise, we are already experiencing soil over-exploitation to keep up with the never-ending demand for food and materials. Consequently, the world's total area of fertile soil is being lost to new habitation and other infrastructures[2]. Thus, as time goes by, humanity requires more soil but has fewer amounts to explore.

Practices like over-using fertilizers and pesticides focus on getting the most out of the soil with little concern for the land's health. Allied to the fact that, according to the Food and Agriculture Organization (FAO), *"generating $3cm$ of topsoil takes 1,000 years"* and *"$12m$ hectares of topsoil are lost every year"*, if intensive farming continues by 2070, arable and productive topsoil will cease to exist [3]. Despite such urgency, some articles refute that *"only 60 harvests left"* is *"overblown"*, but that we undoubtedly *"should not detract from the fact that soil erosion is a problem."*[4]. On the contrary, Our World in Data (OWID) firmly declares that *"6% of soils are estimated to have a lifespan of fewer than 100 years. (...) half have a lifespan greater than 1000 years, and one-third have over 5000 years."*[5].

Soil is also the Earth's layer where flora proliferates. All living beings in this flora group play a major role in the environment as they perform photosynthesis, which is a vital process in a plant's life where it consumes Carbon dioxide (CO2) from its atmosphere, produces energy, and exhales Oxygen (O). From 1982 to 2020, *"global carbon dioxide concentrations in the atmosphere grew about 17%, from 360 parts per million (ppm) to 420 ppm"*[6], and if we do not reduce our global emissions, *"the typical weather*

[1] The World Bank, https://tinyurl.com/vkdvzru7
[2] Overpopulation Environment, https://tinyurl.com/4n2szetn
[3] Soil Degradation, https://tinyurl.com/2p9e9bnf
[4] 60 years left, https://tinyurl.com/267kz2js
[5] OWID 60 left, https://tinyurl.com/5xyrveue
[6] Plants climate change, https://tinyurl.com/y3j8dd6d

patterns will change, some animal species will likely disappear (...) force 100 million people into extreme poverty by 2030" and *"result in the deaths of over 250,000 people globally and annually"*.[7]

As a result, without an accurate evaluation of the soil's properties, such intensive techniques, like applying insecticides excessively and overusing fertilizers, contribute to soil erosion. Maintaining this course of action will undoubtedly lead us to a downfall, where soil transforms into dry land, higher percentages of people starve, and, since soil is a natural solution to reduce greenhouse gases, the unstoppable consequences of climate change.

1.2 Motivation

To protect our soils, it is crucial that new and accessible methods are developed for landowners to avoid over-fertilization. Currently, soil fertility is partially assessed by the concentration of sixteen atoms: C, Hydrogen (H), O, Nitrogen (N), Phosphorus (P), Potassium (K), Calcium (Ca), Magnesium (Mg), Sulfur (S), Boron (B), Copper (Cu), Chlorine (Cl), Iron (Fe), Manganese (Mn), Molybdenum (Mo), and Zinc (Zn). Some of them are minerals, while others are non-minerals, and all are required in different quantities. However, the mere presence of these elements is far from enough to extract the correct amount of chemicals to add. Other soil properties, such as the land's pH level, limit the amount of minerals that the plant's roots can absorb.

There are several methods to estimate these soil characteristics and the number of nutrients that they possess. However, they are still primarily made by conventional laboratory analysis - a reliable but complex process. Interestingly, it is also one of the oldest ways to perform these evaluations, which reflects the saturation in soil analysis investigation. As we can see, in 1970, there were already papers related to chemical soil analysis [12], and in 2021, *"Bray-P1 and Olsen soil tests"*[8] were still advised as the best method for P extraction in Minnesota. This method involves transporting parts of soil to an equipped laboratory, where it is chemically analyzed. Such a process requires access to a laboratory, which is not feasible for most farmers, especially those with limited resources. Therefore, even though it is highly accurate, the method is time-intensive and costly, making it inaccessible to everyone.. Thus comes the practicality of Spectral Vegetation Indices (SVI).

Some of the gases and chemical reactions during photosynthesis respond to wavelengths in the infrared spectrum and can be used to evaluate the plants' well-being. Therefore, SVI are formulas derived from wavelengths dispersed through the reflection spectrum that estimate flora properties and conditions. With the rising number of Earth Observation Satellites (EOS) and the continuous advance in drone technology, the relevance that SVI can have on estimating soil keeps growing. So, unlike laboratory analysis, *"soil sensing approaches such as spectroscopic techniques, (...) offer the opportunity to*

[7] Global warming, https://tinyurl.com/5cp764bf
[8] Minnesota, https://tinyurl.com/nhhfhp59

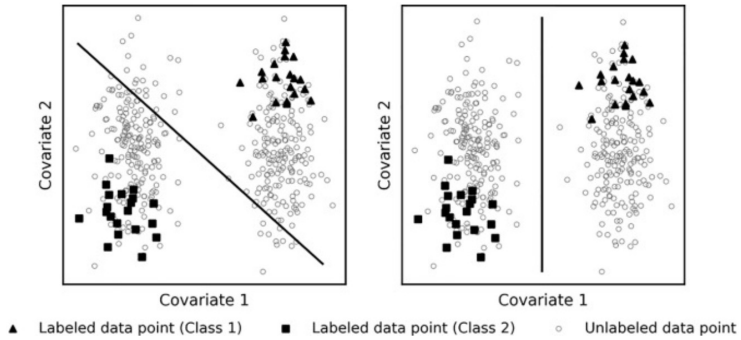


Figure 1.1: An illustration of the usefulness of unlabeled data. Left one shows the optimal classification boundary only based on labeled data, right one shows the optimal boundary with considering both labeled and unlabeled data [1]

measure various soil chemical, physical and biological parameters in a fast and inexpensive way" [13].

Artificial Intelligence (AI) is one of the current most studied and explored areas in computer science, where algorithms are created to imitate the capacity of human brains. Nowadays, it relies on mathematical-statistical approaches that, given a set of data, train a model to predict behaviors - Machine Learning (ML). This area of AI has been subject to several studies in the context of remote sensing. For example, the authors in [14] developed and evaluated a method for predicting soil nutrients using ML. Others also applied ML with remote sensing techniques to estimate soil indicators, like the following articles: [15], [16], and [17].

Unfortunately, most ML algorithms thrive with extensive labeled data sets, and in this field, the percentage of unlabeled data is far larger, as we can observe from the following paper [1], where the author clearly states that there are not enough labels for Supervised Learning (SL) models to train. Such lack might come from the high monetary costs that soil data extraction brings. Truth is, in [6], the author had to create a labeled data set for running soil nutrient estimation ML experiments. Fortunately, there is an emerging technology in ML that can produce accurate predictions by combining a small amount of labeled data with unlabeled data points. This rich area of ML is entitled Semi-Supervised Learning (SSL), where its models are meant to *"understand how combining labeled and unlabeled data may change the learning behavior and design algorithms that take advantage of such a combination"* [18]. Moreover, as simply described in the previously mentioned paper [1], unlabeled data has valuable information and can be used to increase the model's performance 1.1. Thus, as this topic lacks labeled data, exploring SSL methods naturally seems to be the path to follow. Interestingly, there are almost no research articles related to SSL estimating soil nutrient values with remote sensing data.

Python is a powerful, easy-to-write, and commonly used coding option to implement ML models. Its extensive use is also due to the vast amount of potent toolkit - open source software available for python files - made for this specific language, such as Numpy, for fast and reliable matrix calculation and manipulation, Pandas, full of useful data science techniques, Scikit-learn (sklearn), filled with SL

models, and many more.

Even though there is a vast amount of toolkits available, none smoothens the arduous task of extracting EOS data and applying it on ML models. In fact, [6] reported an absence of such capability on contemporary software, and emphasized how relevant such toolkit might be. Therefore, the writer created TerraSenseTK (tstk). It's main objective is to ease the process of creating ML models with EOS data, by covering the whole procedure, from satellite data acquisition, to model fitting. Despite its utility, it missed crucial aspects that prevented its potential, such as lacking a straight-forward interface, code being to specific for the creator's objective, and being partially complex.

1.3 Objectives & Contributions

Taking into account the motivation written above, three equally major objectives rise from this thesis. Firstly, the identification of EOS and soil nutrient datasets that can be used to prepare ML models. Secondly, enhancing the tstk toolkit, by increasing its functionality, clarity and integrating SSL, all without sacrificing performance. Finally, comparing SSL algorithms behaviour in soil nutrient estimation through EOS data with classical SL models. To achieve said goals, the investigation was split into three crucial steps:

1. Discovering, creating, and evaluating EOS datasets.
2. Analysing tstk strengths and weaknesses, implementing or modifying necessary software, and adding ML models.
3. Comparing SSL with SL models using the tstk toolkit.

As it was stated before, ML models are strongly dependent of high-quality data sets. So, in order to accurately compare the performance of SSL and SL, one must guarantee that they are fed with adequate data. It is true that EOS images exist in abundance, however discovering large datasets with soil nutrient quantities has been proved to be a difficult task, as there are a few organizations that supply such information. In fact, such lack forced investigators to create their own datasets, like the author from [6] did. Therefore, by accomplishing said objective, this investigation is aiding the AI community specialized in EOS data, by assessing and consequently forming an opinion of some public datasets, plus creating a high quality one. Ultimately, these findings aid future ML investigators allowing them to focus more of their research time on ML.

Currently, no toolkit is known to have the same potential as tstk, and being such a motivational topic, it is only reasonable to conduct all this investigation inside the same framework as an alternative to testing and discovery of improvements. Having a high quality dataset and a cluster of models to choose from becomes irrelevant, if not properly inserted into the tstk's structure, as it can reach complexity

levels that are hard for the community to grasp and make use of. Therefore, one must analyze and decide the optimal solution to implement these models without compromising its functionality. Moreover, any function that enhances the toolkit's utility is inserted. Both of these additions require an in-depth comprehension of the toolkit, and thus justifying the second step mentioned above. During the tstk thesis [6], the author referenced a statement from National Aeronautics and Space Administration (NASA) presented in a ML workshop in 2020, stating that *"challenges were grouped into three major categories: 1) training data, 2) algorithms and models, and 3) tools and analytic frameworks"*, and underlined that the second challenge is the most crucial, as there is a *"limited number of open-source software and ML frameworks to develop, evaluate and share ML models"*. Therefore, enhancing this software reduces the problem's complexity, and ultimately contributes for the AI community.

Before diving into the third objective, one must expand its knowledge of the features that can be obtained from EOS, and how they correlate to soil nutrient estimation. Such an approach is required in order to select the appropriate formulas to feed into the AI agent and also to discard those that are unnecessary. Therefore, an extensive reading of several articles is crucial to open someone's eyes to additional information and state-of-the-art theories. Several articles regarding the topic include the following [19], and [20]. It is of great importance to mention that this step is not considered as a main objective of our research, but if ignored, it would have compromised the investigation's results.

Lastly, after collecting a series of SSL and SL algorithms and having them correctly implemented in the tstk, tests will be conducted to select the model, or the group of models, that perform best at predicting nutrients when given EOS data. Ultimately, one will be able to evaluate if SSL does indeed surpass SL models when lacking labeled data. From observing their behaviour, which has little to no investigation, this research can additionally be viewed as a pioneer in this topic and thus, represents a major contribution to the AI community specialized in EOS data.

1.4 Document Outline

The remaining of this document is organized as follows: Background provides information about all crucial topics relevant to understand our methodology; Related Works presents important information found in related pieces of literature; Dataset Exploration reports why and how the manipulation process unfolded for the four different public datasets that were found, and demonstrates the creation of a final dataset. Subsequently, TerraSenseTK: Toolkit Improvements displays and meticulously explicates the variety of changes that were implemented into the tstk toolkit. Following this, Semi-Supervised Learning: Case Study Specification elucidates the experiments that will be executed. Subsequently, we present and deliberate upon the results obtained with the previously justified experiments, and conclude this paper in Conclusion by summarizing our findings, recalling our limitations and challenges, and suggestions

of future work.

2

Background

Contents

In this chapter, the reader will find a description of the core concepts required to fully understand the investigation process. It begins with Section 2.1 which overviews soil fertility requirements, then Section 2.2, where the reader is enlightened with EOS data acquisition and interpretation. It terminates with Section 2.3, which includes some crucial ML principles, SL and SSL concepts, relevant Data Science (DS) techniques and all the algorithms that were chosen to conduct our experiments.

2.1 Soil Fertility

As it was stated in the previous section, to assess the algorithms' behaviors and to interpret data sets, one must understand what conditions the soil has to verify to be blooming and fertile. Additionally, comprehending the plant's processes, such as photosynthesis, is useful for perceiving how and which SVIs serve our purpose. Such topics are clarified in the following sections.

2.1.1 Cation and Anion Exchange

Cation Exchange Capacity (CEC) is a measure that represents the capacity of soil to hold on to cations (positively charged ions). In other words, it is the number of anions (negatively charged ions) that a soil possesses. Since anions and cations are strongly attracted to each other, soils with higher values of CEC can capture more quantities of cations [21]. On the other hand, Anion Exchange Capacity (AEC) represents the amount of cations present in the soil, which can be used to measure its capacity to retain anions. They are commonly measured in units of milliequivalents per 100 grams of soil (meq/100 g) [21].

According to [8], these measures are affected by the soil's pH levels. It states that *"at low pH, more positive charge exists due to higher* H_+*"* and that *"as pH increases,* H_+ *concentration decreases, which raises negative charges."*. Therefore, soils with high pH levels have higher CEC, and soils with low pH possess increased AEC. This information stresses the importantance of knowing the pH of the soil before adding fertilizers, as adding high amounts on soils with low CEC causes leaching and environmental pollution. On the other hand, applying few nutrients to soils with high CEC leads to mineral deficiency [21].

2.1.2 Soil Nutrients for Fertility

A crop's healthy growth requires sixteen natural elements, minerals, and non-minerals. C, H, and O, represent the only non-mineral elements that the flora requests. The other thirteen elements *"are taken up by plants only in mineral form from the soil or must be added as fertilizers"* [22]. N, P, K, Ca, Mg, and S are all macro-nutrients, although the first three are referred to as primary elements since they are required in higher quantities than the other three, the secondary elements. The remaining micro-

nutrients consist of B, Cu, Cl, Fe, Mn, Mo, and Zn. These elements occur in very small amounts in soils and plants, but their role is equally important as the primary or secondary nutrients. For this paper, we solemnly focused on N, P and K.

N is the most versatile element required for soil fertility, as it can be used in organic and inorganic form, as a solution and as a gas, as well as a cation and an anion [22]. Plants capture this element primarily through Inorganic nitrate (NO3-) and Ammonium (NH4+) ions embedded in the soil solution. However, N is not a natural constituent of rocks or minerals. Rather, the natural state of N is as N_2 gas in the atmosphere [22]. This element tremendously impacts the plant's health, and one must be careful to maintain it in adequate quantities. As the previously mentioned article shows, *"a good supply of N is associated with vigorous growth and a deep green color"*. Yet, when missing, the flora turns *"stunted and yellow"*. Additionally, its excess *"causes plants to remain in a vegetative growth stage and delay initiation of flowering or fruiting, resulting in lowered yields of some crops"* [22]. As specified by the Colorado State University[1], lands should contain $40\ ppm$ of N. Contrarily, the article [8] states that plants should possess $15000\ ppm$ of N.

P is the succeeding primary element. Found in the soil and mainly extracted from a mineral entitled apatite, but it can also be found in other sources like *"decaying plant and animal residues, humus, and microorganisms"* [22]. It is absorbed through the plant's roots as one of two different anions, $HPO_4{}^{-2}$ or $H_2PO_4{}^{-}$, being the first present in soils containing *"pH values greater than 7.0"*, while the second one *"with pH between 4.3 and 7.0"* [22]. This inorganic substance *"stimulates young root development and earlier fruiting"* by regulating energy and plant growth. Therefore, its absence leads to plant stunt and an *"abnormally dark green color"* [22]. As [8] claims, P quantities should be $2000\ ppm$.

Lastly, in the category of primary elements, there is K. Plants incorporate this mineral as the cation K^+ through the molecule K_2O. Unlike the others, this cation does not have a main purpose. Instead, it serves to aid several parts of the plant. Around *"60 enzymes require the presence of K, with higher concentrations found in the active growing points and immature seeds"*. It also takes part *"in photosynthesis, in carbohydrate transport, in water regulation, and in protein synthesis"* [22]. Thus, a plant with the right amounts of K is invulnerable to diseases, has finer growth, and is tolerant to droughts [22]. Nevertheless, when absent, plants become stunted and develop poor root systems. Such lack can be perceived through the *"bronzing near the edges of lower leaves"* and eventual death [22]. Hence, maintaining a proper amount of K is crucial for crop yield. As stated by [8], K concentrations in the plant must be around $10000\ ppm$.

As stated before, all these elements must be present in the correct amount to contribute to soil fertility and, consequently, the plant's well-being. Table 2.1 summarizes what was previously written.

[1] CSU, https://tinyurl.com/yck3hctx

Nutrient	Concentration (ppm)
Nitrogen	40 - 15000
Phosphorus	2000
Potassium	10000

Table 2.1: Optimal N, P and K nutrient quantities for soil fertility according to [8]

2.2 Satellite Data Acquisition and Interpretation

In this section, the reader may find a combination of concepts mandatory to understand the full EOS data acquisition process, and its potential to predict soil nutrients.

To do so, we begin by presenting fundamental satellite notions in Section 2.2.1, which is followed by Section 2.2.2, where it explains the principles behind remote sensing. Subsequently, Section 2.2.3 introduces several contemporary space missions and important hardware specifications used to acquire land data. Finally, this chapter concludes with Section 2.2.4, where it is demonstrated how satellites can be used to acquire SVI and also how such data is related to nutrient analysis.

2.2.1 Satellite Concepts

An orbit is a regular, repeating path that an object in space takes around another[2]. The one in orbit is called a satellite, which can be classified as natural or artificial. Natural satellites are created by nature, like our moon, whereas artificial ones are made by humankind, like the International Space Station (ISS). Our investigation will consider satellites as man-made objects orbiting around Earth.

There are four types of orbits: Geostationary Orbit (GO), Low Earth Observation Orbit (LEO), Medium Earth Observation Orbit (MEO), and Sun Synchronous Orbit (SSO). They differ in distance from Earth, affecting their velocities and revisiting times. GO satellites are 37,015 km above Earth, constantly overlooking the same spot, resulting in a precise 24-hour revisiting time. LEO satellites are much closer at 161 to 322 km enabling them to capture high-precision images. MEO satellites are slightly further out, between 2,000 km and 35,000 km. SSO satellites are a subset of LEO satellites, orbiting at a 90° angle to the Equator, crossing geographic locations at the same local time each orbit, such as passing over the same town at 5 pm.

Since SSO are close enough to Earth to extract high-quality data, plus as they possess the ability to always capture data from a location at the same time, all datasets subsequently mentioned as EOS data are extracted from SSO satellites.

[2]What is an orbit - NASA, https://tinyurl.com/2yahvhau

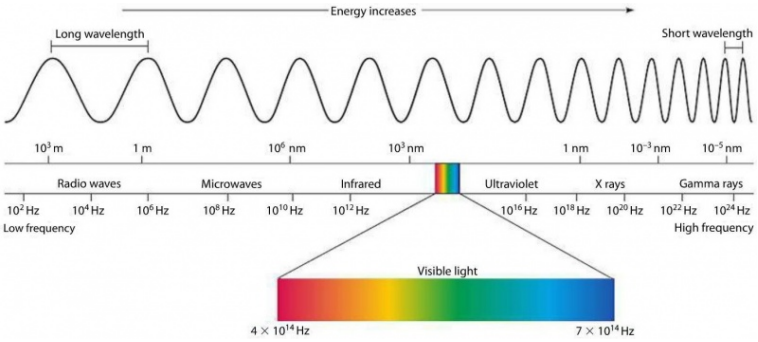


Figure 2.1: EMS with visible light highlighted

2.2.2 Data Acquisition

Satellites, however, are useless if one can not extract data from their sensors. Thus, this section aims to familiarize the reader with the data acquisition process behind any sensor.

Electromagnetic radiation is energy that travels through space at the speed of light. These radiations vary in wavelength and frequency. The higher the energy it transmits, the shorter its wavelength and the higher its frequency. The Electromagnetic Spectrum (EMS) is the range of all types of electromagnetic radiation. It is represented as a line and interpreted from left to right with increasing frequency and decreasing wavelength 2.1[3]. Therefore, it is divided into regions conforming to the ray's properties. Radio waves, with the longest wavelengths, are used for communication, microwaves to cook food, visible light, the only region humans can capture, X-rays for medical imaging, ultraviolet to perform sterilizations, and gamma rays for cancer treatments [6].

Remote sensing refers to the process of collecting information about an object using sensors without establishing physical contact. It can be divided into two types: active and passive. Active remote sensing involves transmitting a signal and measuring the response returned from the signal sent. A medical X-ray scan is one of the numerous examples of active remote sensing. On the contrary, passive remote sensing measures the electromagnetic radiation naturally emitted or reflected by the object under study. Our eyes, or a camera, are examples of passive remote sensing instruments. [6]

Multispectral Instruments (MSI) are sensors or devices designed to measure electromagnetic radiation in multiple wavelengths or spectral bands. Bearing that dissimilar rays reflect different energy values, one can extract more information about the soil using MSI. Thus, EOS that possess such capabilities have higher importance to our investigation [6]. Moreover, to avoid confusion, it is also important to mention that satellites with MSI technology address each one of their sensors as bands, and each band has a resolution and a wavelength interval. The resolution, commonly presented in meters (m), represents the distance that is mapped to each pixel of the image captured, and the wavelength interval specifies the wavelength values of the rays that it can perceive. I.e., when we are faced with an image

[3] EMS, http://tinyurl.com/nhznj3fu

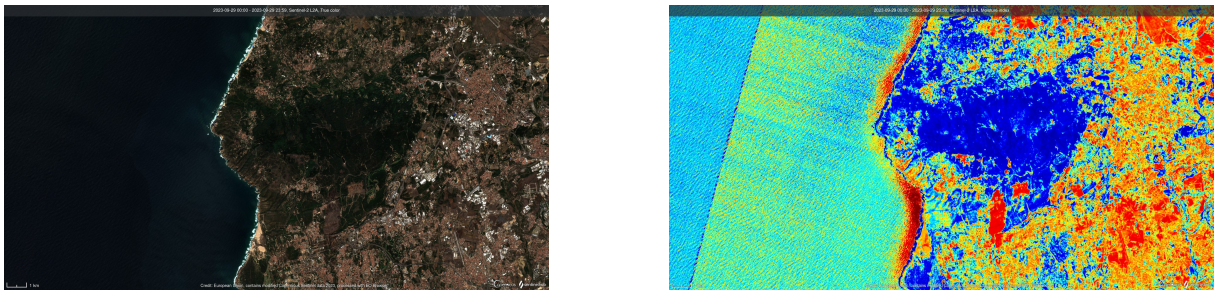


Figure 2.2: Sentinel-2 images of Sintra through RGB (left) and NDMI (right)

captured by a band with a resolution equal to 10 (m) and a wavelength interval of 490 to 510 (nm), it means that each pixel of that image represents the intensity of rays with a wavelength between 490 to 510 (nm), caught in a 10x10 (m) square.

Despite this, visualizing data from a MSI is challenging for the human eye, which can only interpret images from the visible region of the EMS. Human-perceived images are composed of pixels, each with three color values: Red (R), Green (G), and B, known as the Red-Green-Blue (RGB) spectrum, summarizing all observable colors. In contrast, EOS equipped with MSI can capture information in up to 12 different bands, making them invisible to humans. To make these hyper-spectral images perceivable, they undergo a process called false color transformation, where various wavelengths are assigned different colors perceivable by the human eye. Figure 2.2[4] shows false color being applied to an aerial picture of Sintra, a mountainous region close to Lisbon, with a SVI designed to detect moisture.

2.2.3 Space Missions

Nowadays, there are a variety of organizations that possess orbiting EOS with MSI technology, such as the European Space Agency (ESA), NASA, Japan Aerospatial Exploration Agency (JAXA), Indian Space Research Organisation (ISRO), and Chinese National Space Administration (CNSA). Since our study focused on the European and American agencies, the remaining ones are not detailed.

ESA[5] is a space exploration agency from the European Union (EU) that launched several missions to study and analyze the Earth's and outer-space's conditions. From sea level shifts to global winds and the planet's atmospheres to the Earth's ozone layer, ESA is a well-funded and active space agency that is dedicated to researching our world and universe. One of these missions is entitled Copernicus. Launched in 2014, it is an ambitious Earth observation program that aims to provide accurate and timely environmental information for various applications, such as land and agriculture, marine environment, and climate change. This mission contains a set of satellites, named sentinels, for each of these applications. For example, Sentinel-2 (S2)[6] is a set of two satellites, Sentinel-2A and Sentinel-2B, all

[4] 2.2, http://tinyurl.com/wkcn8uav
[5] ESA missions, https://tinyurl.com/2748cctv
[6] S2, https://tinyurl.com/3n5yht5f

multi-spectral and with high-resolution sensors for land monitoring. Its MSI technology is according to the following S2 band information.. Another sentinel group is Sentinel-3 (S3), which is made of several satellites *"prepared to measure sea-surface topography, sea and land-surface temperature, ocean color and land colour with high-end accuracy and reliability"*.[7] Land Use and Coverage Area frame Survey (LUCAS)[8] is an initiative that is conducted by the European Soil Data Centre (ESDAC) [23] to collect soil data across Europe in order to provide detailed and up-to-date information about soil nutrients, bulk density, pH levels, and other properties. Its main purpose is to monitor changes in land use and land cover, and it was specifically made in accordance to the S2 Copernicus mission by selecting soils that these satellites would pass over.

Band	Wavelength Interval (nm)	Resolution (m)
Ultra-blue	430 - 450	60
Blue	450 - 510	10
Green	530 - 590	10
Red	640 - 680	10
VNIR5	690 - 710	20
VNIR6	730 - 750	20
VNIR7	770 - 790	20
VNIR8	780 - 900	10
VNIR8a	860 - 880	20
SWIR9	930 - 960	60
SWIR10	1360 - 1380	60
SWIR11	1570 - 1660	20
SWIR12	2100 - 2290	20

Table 2.2: S2 band information.

NASA has also conducted some Earth observation missions such as Aqua, launched in 2002, with the purpose of investigating our world's water cycle[9]. However, the oldest mission, and most important for our study, is Landsat[10]. It was first deployed in 1972, and its purpose is not only to study the Earth's land cover and land use but also to monitor natural disasters, land use changes, and resource management. Nine constellations participate in this mission, and even though only two are active, all of their data is stored and accessible. In 1972, the constellation Landsat 1-5 MSS (LMSS)[11], in which satellites 1, 2, and 3 took part, was launched. Combined, they stopped working in 1983. Then, in 1982 and 1984, the satellites of constellation Landsat 4-5 TM (LTM)[12] were launched to serve the same purpose, and stopped working in 2013. Following it came the Landsat 7 ETM+ (LETM)[13] that has been active since 1999. However, it had a sensor malfunction in 2003, which compromised the mission.

[7] Copernicus, https://tinyurl.com/2yu9m5f9
[8] LUCAS, https://tinyurl.com/4ktuvm3a
[9] Aqua, https://tinyurl.com/2rp8z96j
[10] Landsat, https://tinyurl.com/36rzevw8
[11] LMSS, https://tinyurl.com/52pbfnd4
[12] LTM, https://tinyurl.com/ycyn8krh
[13] LETM, https://tinyurl.com/wxjybp2r

Finally came the Landsat 8 OLI (LOLI)[14] and Landsat 9 (L9)[15], that were recently deployed in 2013 and 2021, respectively. Additionally, in section Dataset 3: NCSS, the reader may find the purpose for discarding all Landsat satellites besides LTM, which contains the characteristics displayed in Table 2.3.

Band Name	Wavelength Interval (nm)	Resolution (m).
Blue	450 - 520	30
Green	520 - 600	30
Red	630 - 690	30
Near Infrared	760 - 900	30
Shortwave Infrared	1550 - 1750	30
Thermal Infrared	10400 - 12500	120
Shortwave Infrared 2	2080 - 2350	30

Table 2.3: LTM band information.

It is interesting to notice that these organizations have plans to launch other improved EOS for crop observations, which underlines the importance of such measurements. One of these up coming missions is the Chime mission [16], and the Sentinel-4 and Sentinel-5, [17], all from ESA.

2.2.4 Spectral Vegetation Indices

SVI are mathematical formulas that provide information about vegetation's health, biomass, and other characteristics. They correspond to the aggregation of values from different reflected wavelength rays. Several SVI have been developed, each with its specific purpose and assumptions. However, one must understand how bands react to plant and soil properties before rushing into the commonly used SVI for soil health estimation. Only after can the formulas be perceived.

Plants absorb light in the visible spectrum (R, G, and B) for photosynthesis, while they reflect light in the Near-Infra-Red (NIR) spectrum. So, if a passive remote sensor for the RGB spectrum that is pointing to a plant receives low values, then one can conclude that the plant reflects small values of RGB energy, and on the contrary, it absorbs high values. Therefore, it is certain that the plant is receiving enough energy to maintain good health. On the other hand, if it were a NIR sensor, then it would mean the opposite. As the reader will notice further on the document, a total of twenty-eight SVI were considered and applied to our dataset, however, explaining all of them one by one seems unnecessary to grasp their concept. Therefore, being Normalized Difference Vegetation Index (NDVI) one of the most commonly used SVI for soil fertility, it will be served as the explanation.

NDVI uses the reflected values of R and NIR wavelengths to estimate vegetation density. It is sensitive to active photosynthetic compounds and is a popular method used to measure the productivity of

[14] LOLI, https://tinyurl.com/msw3upua
[15] L9, https://tinyurl.com/ycxmrb9n
[16] Chime, https://tinyurl.com/2p9b983e
[17] Sentinel-4/-5, https://tinyurl.com/4bua7jan

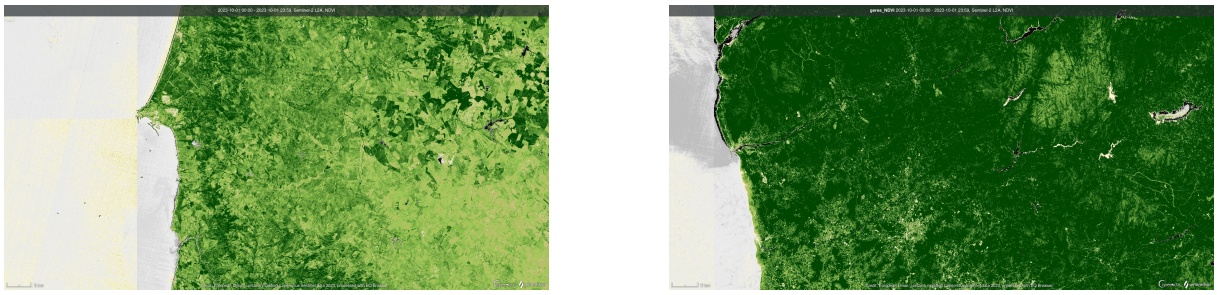


Figure 2.3: Alentejo (left) and Gerês (right) through NDVI

vegetation or *"greenness"* [24]. It has also been used to estimate soil nutrient content, such as N, P, and K. It is calculated according to the formula displayed below (2.1). It ranges between -1 and 1, and values above 0.2 are considered good vegetation indicators [24]. However, as mentioned before, good values depend on the crop in question. This index can be visualized in picture 2.3[18], where one can compare a dry region (left picture) NDVI value with an high vegetation area (on the right), and verify that the values intensify when plant life is blooming.

$$NDVI = \frac{NIR - R}{NIR + R} \tag{2.1}$$

Some of these SVI were already investigated in several articles and proven to be related to nutrient values, whereas others have not. For the purpose of this investigation, it was considered a variety of SVI from the Index database[19] website, despite being partially proven, and, as seen later on, unnecessary.

2.3 Machine Learning

ML consists of a set of techniques for making computers learn and make decisions autonomously, i.e., without being programmed to do so. It is a subset of AI, where computers learn from data and make predictions or decisions for a given input. They analyse the data and create patterns based on their discoveries. For example, a ML agent feed with a large dataset of images labeled as `cat` or `dog` would then learn to classify new images based on patterns and features extracted from the data. Thus, given an input image, it would respond with the class it thinks is the closest to.

2.3.1 Categories of Machine Learning Algorithms

There are several different types of ML, including SL, Unsupervised Learning (UL), and SSL, and all these architectures go through a training and testing step. It is during the first phase that the models

[18] 2.3, http://tinyurl.com/wkcn8uav
[19] Index database, https://www.indexdatabase.de/

analyze and perform calculations transversal to all data points. Here, they gain insight on the data that is required by their core algorithms to perform successful predictions. Upon its finish, it becomes prepared to receive an unknown set of data and to predict each new data point (testing). This phase focuses on assessing the model's performance by comparing its prediction with the correct result.

In SL, the algorithm is trained on labeled data, which means that the data includes both input data and the corresponding correct output, such as the Logistic regression model[20]. On the other hand, UL models are fed only with data points that do not have an association to its correct label. Therefore, they find solutions by discovering patterns and relationships in the data. The Principal Component Analysis (PCA)[21] model is one of the many examples of UL algorithms. Its between these two types that SSL emerged from. It combines supervised and unsupervised learning, as the algorithm trains the model with a small amount of labeled data and a considerably larger quantity of unlabeled data.

In addition to this specification, ML models can also be divided into classifiers or regressors, being that the firsts' purpose is to assign data points to predefined categories or classes, whereas regressors are meant to predict continuous values. Simply said, one would create a classifier to predict if a certain soil is fertile or infertile given it's properties, but would need a regressor if it were required to calculate the quantity of P it contains. Furthermore it will be explained with precise detail why this research deviated to classification and not regression, which is ignored for the remaining of the document.

2.3.2 Supervised Learning Models

From the enormous variety of classification models, we made use of five different algorithms, them being the K-Nearest Neighbours (KNN) algorithm, Complement Naïve Bayes (CNB), Random Forest (RF), Adaptive Boosting (AdaBoost) and finally Gradient Boosting (GradBoost). Their behaviour and logic are explained on the remaining of this subsection.

The KNN classifier is a simple yet powerful algorithm that represents all data points into a feature space, where new data that shares the same features can be directly compared with the whole dataset. During the prediction phase, it calculates the distance between a test data point and all points in the training dataset, typically using Euclidean distance (2.2). It then selects a pre-defined number of nearest neighbours, which are the k training data points with the lowest distance to the test data point. Then, the algorithm assigns the class with highest representation on the neighbours group. While KNN is intuitive and easy to understand, its computational cost increases with the size of the training dataset, and its effectiveness can be sensitive to the choice of distance metric and k value.

$$Euclidean\_distance(X, Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.2}$$

[20] Logistic regression, https://tinyurl.com/34jejr56
[21] PCA, https://tinyurl.com/dd33u8c9

Following this, the Naïve Bayes (NB) classifier is a probabilistic algorithm based on Bayes' theorem. It calculates the probability of each class label given the input features using the following formula (2.3), which does not take into account the presence or absence of other features. The classifier then selects the class with the highest posterior probability ($P(Y|X)$) as the predicted label. Despite its simplicity and efficiency, NB is highly dependent on the feature independence assumption, and is extremely vulnerable to unbalanced datasets. On the contrary, CNB considers the complement of the feature counts for each class. This means that instead of modeling the likelihood of each feature occurring in a class, CNB models the likelihood of each feature not occurring in a class.

$$Bayes\ theorem : P(Y|X) = \frac{P(X|Y) \times P(Y)}{P(X)} \tag{2.3}$$

The RF classifier is a versatile and powerful ensemble learning algorithm that operates by firstly constructing a multitude of decision trees, and then outputs the mode of the class for each individual tree. Each decision tree in the forest is trained on a random subset of the training data and a random subset of the features. During the prediction phase, the RF aggregates the predictions of all individual trees to make the final prediction. Therefore, these algorithms can handle high-dimensional datasets and are robust to noise and outliers. However, they may suffer from high computational cost. Overall, RF are widely used in practice across various domains due to their robustness and effectiveness in handling complex classification tasks.

Regarding AdaBoost, it is a powerful ensemble learning algorithm that combines multiple weak learners to create a strong classifier. During training, AdaBoost sequentially fits a series of weak learners, such as decision trees with limited depth, to the dataset, where each focuses on the instances that the previous models misclassified, assigning them higher weights to prioritize their correct classification in the next iteration. In this way, subsequent weak learners learn to correct the errors made by the previous ones, gradually improving the overall model's performance. When this process is concluded, AdaBoost aggregates the predictions of all weak learners using weighted majority voting to make the final prediction.

GradBoost is an ensemble learning technique that builds a strong predictive model by sequentially training a series of weak learners, typically decision trees, to correct the errors of the preceding models. During training, each subsequent weak learner focuses on the residual errors of the previous ones, gradually reducing the overall error of the model. The final model aggregates the predictions of all weak learners to output a final result. GradBoost is known for its high predictive accuracy and robustness against overfitting. However, it may require careful tuning of hyperparameters and can be computationally expensive, especially when dealing with large datasets.

2.3.3 Semi-Supervised Learning

Since part of this thesis is focused on investigating the advantages SSL might have over SL when large numbers of unlabeled data are present, this section is meant to enlighten the reader with core concepts and costumes that come with SSL methods. Moreover, to avoid confusion and simplify interpretation, X will be used to represent the features dataset, Y all labels for each data point, X_l features of all labeled data points in X, and X_u features of all unlabeled data also in X. The lower-cased version of these symbols represents one single point of the set.

Before learning about some contemporary SSL areas of study, the reader must bear in mind that, in spite of being a powerful concept, these models are truly dependent on one of four key assumptions, which, when not present, reduce their advantage on typical SL models. According to this paper [2], they are:

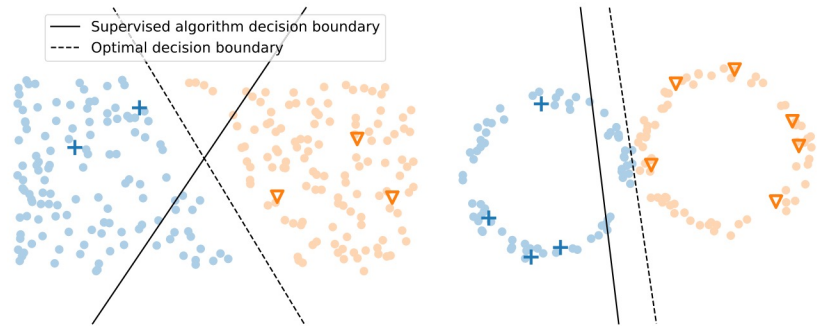
1. **Cluster Assumption**: The cluster assumption suggests that data points with similar features will likely have the same labels. It assumes that the X_l and X_u share a similar data distribution, meaning that the structure of the data, particularly the decision boundaries between classes, should be consistent. This assumption helps in propagating information.

2. **Smoothness Assumption**: The smoothness assumption posits that the decision boundary between different classes should change gradually as one moves through the feature space. In other words, nearby data points are likelier to have the same labels as distant data points. This assumption allows SSL algorithms to generalize from X_l to X_u by leveraging the underlying smoothness of the data distribution. This assumption can be visualized in SSL assumptions according to [2].

3. **Low-Density Separation Assumption**: This assumption assumes that decision boundaries tend to pass through low-density regions of the data, where data points are sparsely distributed. This is because the data points in dense regions are usually easier to label and less uncertain. This assumption can also be visualized in SSL assumptions according to [2].

4. **Manifold Assumption**: The manifold assumption suggests that high-dimensional data often lies on a lower-dimensional manifold within the feature space. It is assumed that the manifold structure of the data is preserved between X_l and X_u. Like the two topics mentioned above, this presupposition may also be previewed in SSL assumptions according to [2].

Following [2], and as portrayed in Types of SSL according to [2], SSL models can be divided into two major types: inductive and transductive. The first one creates models that predict labels for X_u and to any unseen data - data not included in X. Contrarily, the second one is limited to X_u prediction. According to [2] nomenclature, inductive models are split into three categories: Wrappers, Unsupervised pre-processing and Intrinsically semi-supervised.



(a) Smoothness and low-density assumptions. **(b)** Manifold assumption.

Figure 2.4: SSL assumptions according to [2]

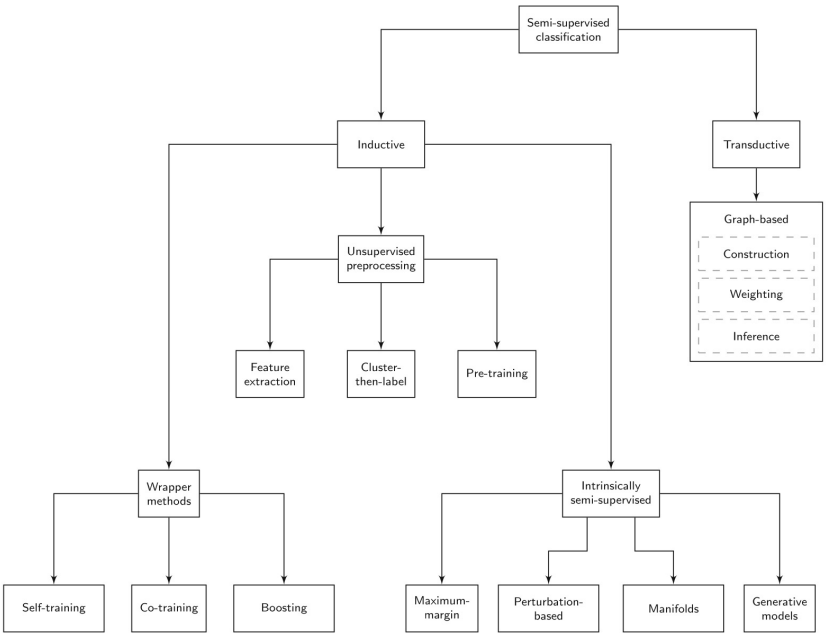


Figure 2.5: Types of SSL according to [2]

Unsupervised pre-processing utilizes several well-known UL models to transform all X_u into X_l, thus transforming the problem into SL. On the other hand, Intrinsically semi-supervised models *"incorporate unlabelled data into the objective function or optimization procedure of the learning method."* [2]. So, they represent all models that were specifically created for SSL. The remaining sub-type of inductive methods, named *"Wrapper methods"* [2], train on X_l, predict labels in X_u, and finally increment these *"pseudo-labels"* into the X_l for model re-training. This process is described in paper [2], where *"unlabelled data is pseudo-labelled by a wrapper procedure, and a purely supervised learning algorithm, unaware of the distinction between originally labelled and pseudo-labelled data, constructs the final inductive classifier"*. This approach becomes extremely helpful as it *"can be applied to any given supervised base learner, allowing unlabelled data to be introduced in a straightforward manner"* [2].

Despite the potential behind the first two categories described, in order to maintain a clear line of thought, and since they were not the subject of study, the document only deepens into Wrappers. Starting from the simplest Wrapper, self-learning is an inductive technique that does not deviate too much from SL models. Self-learning describes SSL models that train on the X_l and make predictions on X_u. Then, they choose a certain amount of x_u with the highest confidence scores and insert them in X_l. This process is called pseudo-labeling and is done through a max probability process, where the model assigns the label with the highest probability. These models vary on the methods use to select the x_u points used as pseudo-labels. One can have a threshold that specifies the minimum confidence value for pseudo-labeling, another can rely on the k x_u data points with higher confidence.

Another wrapper technique is Co-training, which leverages multiple distinct views of data to enhance ML model performance. The core idea is to use multiple information sources, each as a view, to provide complementary insights. These views are fed into different models, partially representing X. The process starts by splitting X into K parts (one for each model) and training each model with its view to predict X_u. The most confident predictions are shared with the other models. Some algorithms split data into features, others into data points. Note that this method is ineffective if the views alone cannot predict the labels, so each view must be related to the labels.

Ensemblers include all models that, as explained in [2], *"consist of multiple base classifiers, which are trained and then used to form combined predictions"*. Bagging and boosting are two diverged branches that emerge from this simplistic approach. The first corresponds to a combination of independent models trained with different subsets of X. The final prediction is made taking into account each model's prediction. With boosting however, *"each base learner is dependent on the previous base learners"* [2], as all x come with a weight value, that is higher if incorrectly classified by the previous base learner. The final prediction, as written in the same paper, *"is obtained as a linear combination of the predictions of the base classifiers"*. So, bearing in mind that bagging results in an agglomeration of independent models, the only truly SSL adaptation for it would result in a combination of Self-training models [2]. *"On the other hand"* boosting has *"an inherent dependency between base learners"*, and so *"can be readily extended to the semi-supervised setting, by introducing pseudo-labelled data after each learning step"* [2].

2.3.4 Model Evaluation

Whereas the prior section reports SSL models' structure, this section presents another crucial part for creating an accurate ML model. The training and testing procedures.

Training a ML model involves using a subset of X, mentioned as X_{train}, to learn the input data parameters. During this process, the algorithms undergo specific modifications to learn the hidden patterns behind the data. For example, Deep Learning (DL) models reduce/increase each of their layers'

weights, or Naive Bayes models compute the probability for each class. On the other hand, testing requires evaluating the performance of the trained model on another subset of X, called X_{test}, that was not used during the training process. It is important to mentioned that, for the models to be evaluated, independently of being SL, SSL, or UL, the test subset (X_{test}) must only contain targeted data. I.e, all x_{test} data points have a target value (y_{test}). So, after finishing its training, the model predicts a target for each x_{test}, which is then compared to the corresponding y_{test}.

Currently, ML engineers mainly make use of two methods to split datasets into training and testing subsets: Random splitting, and Cross-validation. With Random splitting, a chosen percentage of random data points from X are selected and used to create X_{train}. Even though it is a straight forward method, there is a lack of consensus on the specific percentage value to use, only that the training size must be between 70% and 80%: *"Empirical studies show that the best results are obtained if we use 20-30% of the data for testing, and the remaining 70-80% of the data for training"* [25]. Cross-validation however, is a pivotal methodology in the field of ML, essential for assessing the general performance of predictive models. Its primary purpose is to mitigate the problems that arise with small datasets, by partitioning it into subsets. The most common form, K-Fold Cross-Validation, subdivides the data into K parts, employing one as a validation set and the remainder for training in a repeated manner, ultimately yielding an averaged performance metric. This technique aids in model selection and provides a robust estimate of a model's true predictive capabilities, enhancing its reliability and applicability in real-world scenarios.

In a classification problem, each label prediction is accounted as a True Positive (TP), True Negative (TN), False Positive (FP), or a False Negative (FN), where True implies that the model's prediction is equal to the target value, and False the contrary. Positive or Negative corresponds to the predicted class, being Positive the class translated to the problem's solution. So, considering the case where a model was trained to identify if an image is a cat, the Positive class becomes undoubtedly `Cat` and the Negative class `not-Cat`. Thus, a TP corresponds to the model receiving a cat image and classifying it as such, and TN would be the model predicting `not cat` when fed a dog picture. On the other hand, a FP would be the model stating that a non-cat image is a cat and a FN means that the model predicted non-cat to a cat. These measurements are easily perceived through Figure 2.6[22], which is frequently used when approaching classification problems. Four measurements make use of these terminology to evaluate the models performances, them being accuracy, precision, recall and f1-score, all represented in Equation (2.4), Equation (2.5), Equation (2.6) and Equation (2.7), respectively. Translated to speech, accuracy represents the probability of answering right, precision is the percentage of correctly positive predictions in all positive predictions, recall embodies the model's capability to rightly predict a positive class bearing in mind all positive examples and f1-score general performance.

[22] Confusion Matrix, `http://tinyurl.com/ypjh8ray`

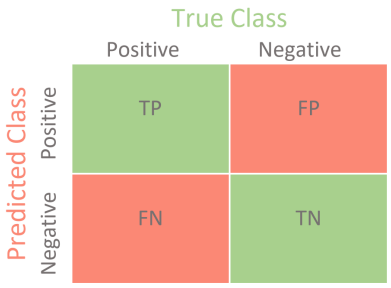


Figure 2.6: Confusion Matrix.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

$$F1\_Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.7}$$

2.3.5 Data Science

Data Science is an interdisciplinary field that harnesses statistical, computational, and domain-specific knowledge to uncover insights and knowledge from vast and complex datasets. It integrates diverse techniques, tools, and methodologies that extract valuable information from raw data. Harnessing its potential reveals patterns, trends, and predictive relationships in the data that would otherwise remain hidden.

Data profiling is an essential preliminary phase in data analysis, encompassing a comprehensive examination of a dataset's characteristics and quality [26]. This process involves assessing the data structure, identifying missing values, detecting data types, and gaining insights into the feature's distribution. Data profiling is the foundation for subsequent data preparation and analysis tasks, ensuring that data is clean, consistent, and appropriately structured for meaningful exploration and modeling. It is during this primordial phase that the dataset is investigated for outliers. They are data points that deviate significantly from most data, or as clearly stated in [27] *"An observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data"*. Identifying and addressing outliers is vital as they can introduce bias and distort the results of statistical analyses and ML models.

There are several methods used to identify outliers in a dataset, but it seems outside of this paper's scope to use another besides the common Interquartile Range (IQR) method. It involves calculating the first quartile ($Q1$) and the third quartile ($Q3$) of the data, which represent the 25% and 75% value of the dataset when ordered from lowest to biggest, respectively. The IQR is then computed as the difference between $Q3$ and $Q1$ and all data points that fall into the group calculated according to 2.8 are considered to be outliers.

$$\{x \in X : x < Q1 - 1.5 \times IQR\} \cup \{x \in X : x > Q3 + 1.5 \times IQR\} \tag{2.8}$$

Feature selection is a crucial step on ML experiments that, when used, comes after data profiling. Its purpose is to select the most important features and discard all those that were considered to be unnecessary, either because they repeat the same information as other features, or as they have no connection to the target. Such process has an enormous impact, since reducing the dataset's dimensionality by removing irrelevant data increases the model's execution time, and increases the algorithm's accuracy. There are several algorithms developed to conduct feature selection, but in this research, only three were taken into consideration. All of them are explained below.

Correlation is a statistical measure that quantifies how two variables change together, assessing the strength and direction of their relationship. The `Pearson correlation coefficient` ranges from -1 to 1, where -1 indicates a perfect negative linear relationship, 1 a perfect positive linear relationship, and 0 no relationship at all. Secondly, Recursive Feature Elimination (RFE) systematically identifies the most significant features in a dataset by sequentially training models and eliminating the least important. This process repeats until a predetermined number of features is reached or until model performance no longer improves significantly. Finally, PCA is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving most variability. It identifies orthogonal vectors representing directions of maximum variance, called principal components, which capture decreasing amounts of variance. By keeping only the components that capture most variance, PCA reduces dimensionality and simplifies data for visualization and speeds up computation.

2.3.6 Hyper-parameter tuning

Subtopics within SSL and broader ML fields have distinct models with unique structures and characteristics. Each of these models divides into parameters and hyperparameters. Parameters are variables updated by the algorithm and derive their values from the data. In contrast, hyperparameters are set manually before training and guide the model's learning process[23]. However, hyperparameters *"do not learn their values from data. We need to manually specify them before training the model."*23, which

[23], Parameters Vs Hyperparameters, `http://tinyurl.com/3pkk9sve`

are then used by the model to update its parameters. For example, a Multi-layer Perceptron can have a learning rate varying from 0.0 to 1.0 and one of several activation functions. With just these two parameters, an infinite number of models with the same structure can be created, yielding different results for the same input X. Therefore, hyperparameter tuning is commonly performed to find the best parameters for the model before testing. This crucial process involves running training and testing for numerous parameter combinations and selecting the ones that achieve the best performance.

3

Related Works

Contents

In the following chapter, the reader may find several contemporary technologies and related datasets that were considered in the investigation process. Throughout the three upcoming sections, we introduce important datasets that are related to the topic, present some state-of-the-art SSL models, and finalize with Python toolkits crucial to this research.

3.1 Datasets

Over the course of the investigation, four major datasets were discovered and analyzed to understand if they had the necessary qualities to build a final dataset. They are mentioned throughout this section.

3.1.1 AfSis Dataset

The Africa Soil information Service (AfSis) dataset was created to provide detailed soil nutrient maps for sub-Saharan Africa in order to enhance agricultural productivity. It was built from the predictions of two ML models (random forest and gradient boosting) that were trained with an agglomeration of soil nutrient data delivered by eight organizations [3].

The AfSis group, which contributed with the most amount of data points, and thus justifying the dataset's name, totalling 78.000 samples spread through 28.100 locations of 10 by 10 km; Ethiopia Soil Information Service (EthioSis) that offered 15.000 samples from Ethiopia, the Nigeria Soil Information Service (NiSis) with 3.000 samples; the International Fertilizer Development Center (IFDC) that contributed with 3.500 samples from Uganda, Rwanda, and Burundi; the One Acre Fund (OAF) with 2.400 samples also in Uganda and Kenya; the University of California (UC) with a small amount of 1.800 samples in Kenya, and finally, VitalSigns with 1.374 samples in Ghana, Rwanda, Tanzania, and Uganda. Due to these contributions, the AfSis database totaled an astonishing 105.074 labeled data points. As a result, the models predicted parts per million (ppm) values in a 250 m resolution for fourteen nutrients in 30 cm deep. Despite this, as reported in [3], their generative models scored R-squared values between 0.1 and 0.86, depending on the nutrients they were estimating. The following graphs display the values it got when predicting three of the macro-nutrients (mentioned in 2.1.2) N, P and K 3.1.

Despite its success, limitations arise from the 250m resolution, as the authors are insinuating that the nutrient ppm values are consistent in an area of 62.500 m^2 $(250(m) \times 250(m))$, which seems unlikely. In fact, to overcome such constraint, they released a similar paper four years after (2021), where they mapped sub-Saharan African nutrient values to 30m [4], by using the S2 satellite. This improved dataset revealed to have increased performance, reaching R-squared values of 0.732, 0.486 and 0.773 when predicting N, P and K. This gain can be visualized in graphs 3.2. Additional problems rise with the spatial clustering of sample locations, resulting in the under-representation of certain regions and land cover types. In fact, as sub-Saharan Africa does not represent vast land cover possibilities, such as tropical

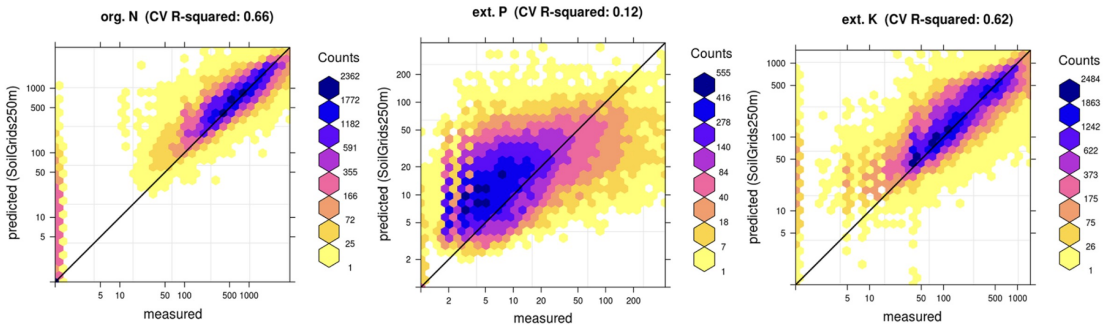


Figure 3.1: R-squared values for organic N and extractable P and K [3].

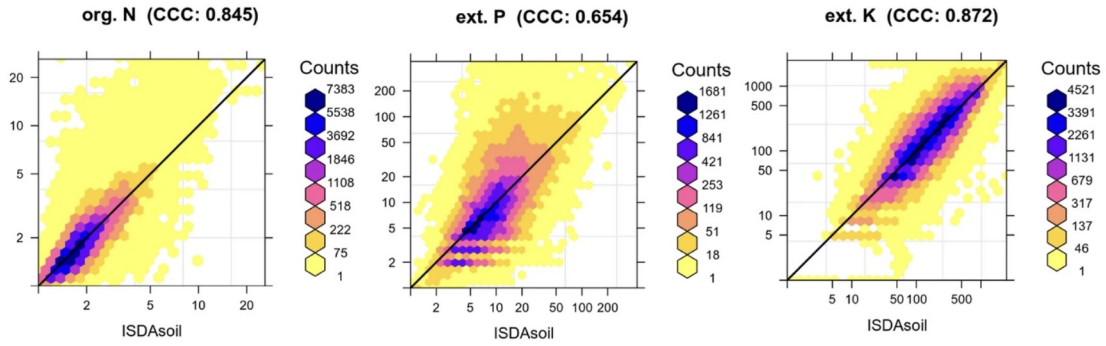


Figure 3.2: Concordance Correlation Coefficient (CCC) values for organic N and extractable P and K [4]

forests or mountainous regions, it should not be used to train models predicting nutrients of any other land type except those in Africa. As the author wrote: "the dataset heavily under-represents tropical jungles or similar remote areas" [4]. Finally, one must not forget that this sub-Saharan dataset is not a real-world dataset and merely values generated from two ML models.

3.1.2 Big Earth Net Dataset

The Big Earth Net (BEN) dataset is a vast and diverse collection of 590.326 non-overlapping image patches from the S2 satellite. Acquired between June 2017 and May 2018, and with corresponding land cover labels, BEN was created with the primary goal of supporting remote sensing and Earth observation research by providing high-resolution data for land cover classification and land use analysis [28].

The dataset was developed by a team of researchers at the Remote Sensing Image Analysis (RSiM) Group and the Database Systems and Information Management (DIMA) group at the Technische Universität Berlin (TU-B). The EOS data was obtained from Austria, Belgium, Finland, Ireland, Kosovo, Lithuania, Luxembourg, Portugal, Serbia, and Switzerland. Each image patch was annotated by the multiple land cover classes (i.e., multi-labels) provided by the CORINE Land Cover database of 2018 [29]. As it contains data covering various geographical regions and land cover types, it is a valuable dataset for ML and environmental monitoring applications. However, like many datasets, BEN has some limitations,

including inconsistent labeling and potential cloud cover in the images, which can affect its usability for certain applications. Despite these limitations, it remains a valuable resource for the remote sensing community and environmental researchers, and therefore, selected as one of the datasets to take part in this investigation.

3.1.3 National Cooperative Soil Survey Dataset

The National Cooperative Soil Survey (NCSS)[1] Soil Characterization Database[2] is a comprehensive soil laboratory database built and maintained by the Kellogg Soil Survey Laboratory. It contains information about soil characteristics, including nutrient levels, for various locations across the United States [30].

The dataset is a comprehensive and collaborative effort of the United States to map, classify, and describe soil resources across the country. It was created to address the need for detailed soil information to support agriculture, land use planning, environmental management, and other applications. The project originated in the early 20th century, 1899 to be exact, and it is still ongoing.

NCSS has been used to train a model that predicts soil bulk density with a Random-Forest algorithm [31], where it scored an interesting Root Mean Squared Error (RMSE) of $0.13 g.cm^{-3}$. However, even though it had good results, it also has its limitations, including potential inaccuracies in soil data due to variations in data collection methodologies, changes over time, and the project's large scale. Nonetheless, the NCSS dataset remains an invaluable resource for soil scientists, farmers, land planners, and environmental researchers for soil-related studies and applications.

3.1.4 LUCAS Topsoil & LUCAS Copernicus

In this subsection, the reader may find two datasets constructed by the same organization, ESDAC, with divergent characteristics but similar objectives: to gather information across European soils.

LUCAS Topsoil [32] is a recurrent ground survey initiative conducted throughout the EU countries, with survey rounds in 2009/2012, 2015, and 2018, that gathers soil properties and studies them. As the authors wrote on the latest paper, *"it presents an overview of the laboratory analysis and describes the spatial variability of soil properties by land cover (LC) class and a comparative analysis of the soil properties"* [32]. According to their statement, 2018 LUCAS Topsoil contains 18.984 locations with soil properties from 0 to 20 cm deep. The other dataset is named LUCAS Copernicus [33]. Even though also being a periodic program with previous completions in 2006, 2009, 2012, 2015, and 2018, contrary to Topsoil, it is specifically designed for EOS purposes. It focuses on surveying land cover extending up to 51 meters in four cardinal directions around each observation point. The paper [33] outlines the LUCAS Copernicus protocol and ensures consistent land cover data collection for an area of up to 0.52

[1] NCSS, https://tinyurl.com/36t8xrjc
[2] NCSS-database, https://tinyurl.com/37ujj49m

hectares. Additionally, it presents a methodology to generate a high-resolution dataset suitable for EOS land cover and land use applications, utilizing satellite imagery. The survey covered a total of 63.364 points, spread across 26 land cover classes. Using this information, polygons were delineated for each point, resulting in 58.428 polygons providing land cover data, including 66 specific classes such as crop types, and information on land use encompassing 38 different categories, which were inherited from the LUCAS core observations. This dataset is valuable for various EOS applications and land management purposes.

Like the other three mentioned datasets, there are some limitations, such as potential variations in data quality and resolution due to the diversity of national survey methodologies. Nevertheless, these LUCAS datasets are a vital resource for land management, environmental monitoring, and ML training for nutrient prediction.

3.2 Semi-Supervised Learning

In this section, the reader may find SSL models that were investigated for research with similar objectives to soil nutrient estimation through EOS data. Additionally, in the following sub-section, one can review some papers that made use of Self-training models.

3.2.1 SSL research

This technology was first mentioned in a 1989 where the author began to investigate how one can learn from unlabeled data, by trying to answer *"whether it is possible to learn (...) without a teacher labeling examples of each concept to be learned as positive or negative"* [34]. Some years later, in 1995, another author introduced self-training algorithm for sense disambiguation, rivalling its contemporary SL alternative by scoring 96% accuracy [35], and in 1998 the authors of [36] presented the Co-training model for web page classification, which also got better results then another SL models. Later on, in a 2015 paper [37], a study was accomplished on several self-training techniques and how they performed given different quantities of X_u and X_l. Only in 2020 [5] [2], and 2022 [38] three surveys were written mentioning new models that were adapted, or created, to serve SSL. In these papers, no investigation was conducted, as one of the authors mentioned: *"our survey constitutes an up-to-date review of this important topic within machine learning"* [2]. Combined, they serve as a crucial source of knowledge for SSL, by extensively describing several architectures that one can follow to make use of X_u. Additionally, they clearly state that their models can utilize X_u to increase their accuracy, as one can perceive from the following picture: 3.3

To the best of our knowledge, there are no papers where the authors applied SSL models to predict soil nutrient quantities. It is unknown why this area of ML has not yet been entirely investigated, but it

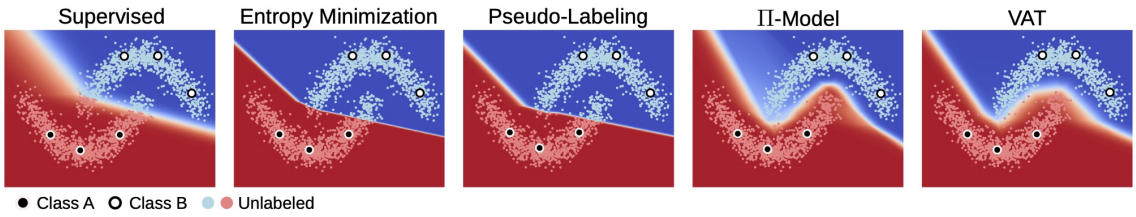


Figure 3.3: The decision boundaries obtained on two-moons dataset, with a supervised and different SSL approaches using 6 labeled examples, 3 for each class, and the rest of the points as unlabeled data [5].

is suspected that it is due to the heavy computational resources that DL models require, plus the lack of datasets. In fact, nearly all authors investigated the behavior of SL models, such as Support Vector Machine (SVM), RF, Multiple Linear Regression (MLR), and either diverged on the Earth's soil location to estimate or the properties to predict. Such a lack elevates the investigation's relevance, but prevents a final comparison. Consequently, other papers with SSL investigations were considered in this review.

In [39], the authors reinforce that DL algorithms have surpassed state-of-the-art methods to classify land use in high-resolution remote sensing images. However, they underline their necessity for labeled data, which is scarce in this field. Thus, they created a SSL framework that combines DL features and self-label techniques. Impressively, after being experimented with four different datasets with a small number of labels, it scored an average accuracy of 92%. Additionally, another paper [40] reinforced the SSL strength in little labeled datasets. The investigation aimed to classify Sea-Ice on polar satellite images, where they created a SSL Generative Adversarial Network. After its implementation, they compared it with a SL model. They concluded that *"SSL improves the overall accuracy achieved by the SL approach by at least 5% in configurations with less than 100 labeled samples"*. In summary, both papers clearly state that when labels are scarce, SSL performs better.

3.2.2 State of the Art

From the three papers mentioned above, [2] [38] and [5], one has a considerable amount of SSL models to explore. However, this research was narrowed just to Self-training to maintain a direct comparison between SL and SSL. Ultimately, if Self-training models surpass their SL versions, one may theorize that other SSL models purely created for these problems may exceed astronomically. Therefore, in this subsection the reader will not find a full description of the diverse amount of SSL models presented on those papers, but other Self-training investigations. It is important to stress that many, if not all, SSL models presented in [2], [5], and [38] are merely theoretical, as they are not only missing investigation but also secure implementations, which inevitable raises uncertainties in our research topic.

In paper [1], the author studied the divergence between SL and Self-training models when classifying soil. Their dataset was an agglomeration of topographic points in a region of China that was mostly *"formed on deposits of silt loam loess, except for the valley, where the underlying parent material is*

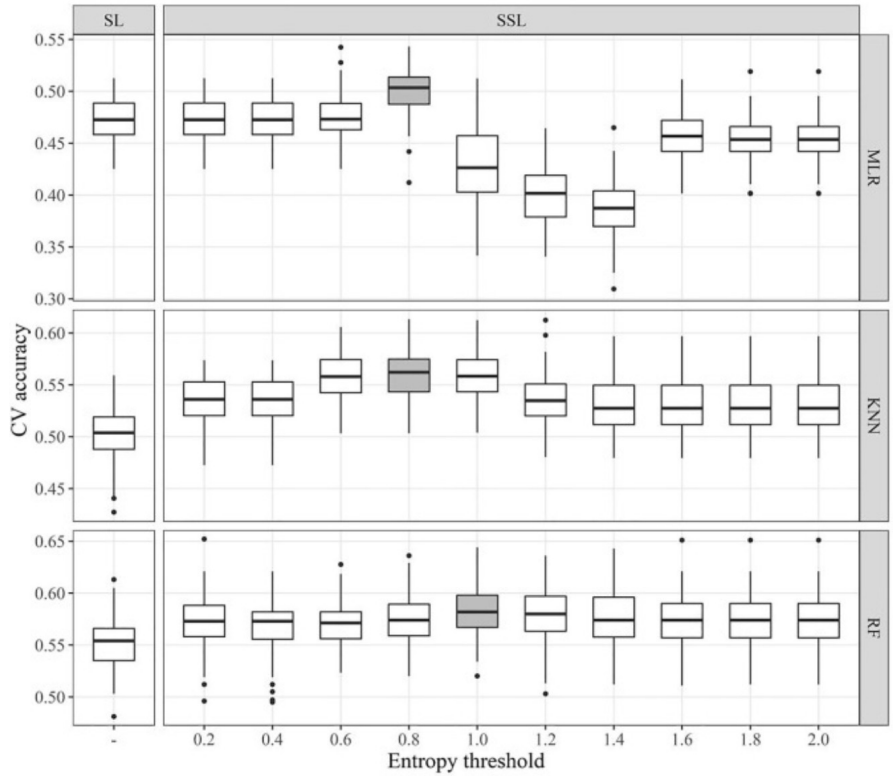


Figure 3.4: Boxplots of cross-validation (CV) accuracies based on three models (multinomial logistic regression (MLR), k-nearest neighbor (KNN) and random forest (RF)) with the supervised learning (SL) method and the semi-supervised learning (SSL) method over different entropy thresholds, boxes marked in gray color for SSL represent the highest average CV accuracy at that threshold value. [1].

fluvial deposits" [1]. Additionally, the area is mainly used for croplands, which are limited to soybean or wheat, nothing more. In order to solely evaluate the gain that an SSL approach might give, the authors used the same models for SL and Self-training. They were the MLR, KNN and RF. The dataset was built from 129 labeled samples (X_l) and 10.000 randomly selected X_u from a total of 596.158 points. After conducting their experiments, they plotted their accuracy scores in a box-plot graph 3.4, with which concluded that SSL had higher performance than SL. Nevertheless, it is important to notice that they have poor accuracies (below 0.6) even though having little to none data diversification, and that the experiences used few amounts of X_l.

In another paper [14], the author investigated if SL could predict soil nutrient values from electromagnetic waves *"within the UV-VIS range, i.e., (200–11,000) nm"* [14]. To do so, the research collected 350 soil samples from a local farm in Slovenia, with which formed a *Global and Local soil dataset* [14]. Additionally, this research turned to classification by associating a label for intervals of ppm, which where made with 3 categories: I, II and III. The second dataset split the ppm values into 5 labels. After running a collection of 6 SL models through these 3 categories, the author reported precision values above 80% with Artificial Neural Network (ANN) and Least-Square Support Vector Machine (LS-SVM) models,

when classifying according to the II category. Furthermore, the author also revealed that *"The more soil components such as texture (...) are involved in the classification, the better the soil properties can be predicted"* [14], and concludes with a final experiment that reveals a performance gain in all 6 models when PCA is executed first. So, the paper shows that SL models nearly achieve perfect performance in classifying soil nutrient values from remote sensing data. It also recommends using PCA on the dataset first, though its effectiveness is uncertain due to the small dataset size.

3.3 Libraries and Toolkits

In the realm of modern research and development, the power of libraries and toolkits in shaping the landscape of innovation cannot be overstated. They play a pivotal role in facilitating the creation and advancement of various fields, offering a wealth of pre-built resources, functionalities, and frameworks that can significantly accelerate the development process. Toolkits are an agglomeration of Python files that can be freely downloaded and explored. They are built to increase productivity, as the code is made to be correct, safe, optimized, and straightforward, allowing the developers to focus on their main tasks. In this chapter, the reader can delve into the main Python toolkit that was crucial for this research: tstk. Additionally, three other toolkits are superficially presented, since two of them were investigated to add SSL models, but eventually not used to preserve software compatibility, and the other one was already used to implement all tstk's ML models.

3.3.1 TerraSenseTK

Previously mentioned in chapter Datasets, the author of [6] developed a toolkit that allowed any Python user to easily extract EOS data and explore how some SL models performed with it. tstk, as the author wrote, *"must be capable of reading a dataset, indicating the features the user wants to predict (...) and measuring the estimation."* [6]. Its structure follows the figure 3.5, being made out of 4 Python models: Dataset, Algorithms, Performance and Experiment. Firstly, Dataset was built to download EOS, and to represent the data created according to the user's specification. Secondly, the Algorithms module wraps all Python files with classes representing a predictive ML model, or some other useful ML algorithm. The Performance module is used to encapsulate a series of classes useful for estimating a model's performance and finally, the Experiment module is *"responsible for the execution of an (...) experiment* [6] from which users can select models and other algorithms to include in the workflow. At the moment, as one of the main objectives of this research is enhancing the tstk toolkit functionality, it is important to mention that it lacks validation from other sources besides the author's, and that it is thoroughly detailed in the upcoming section 5.

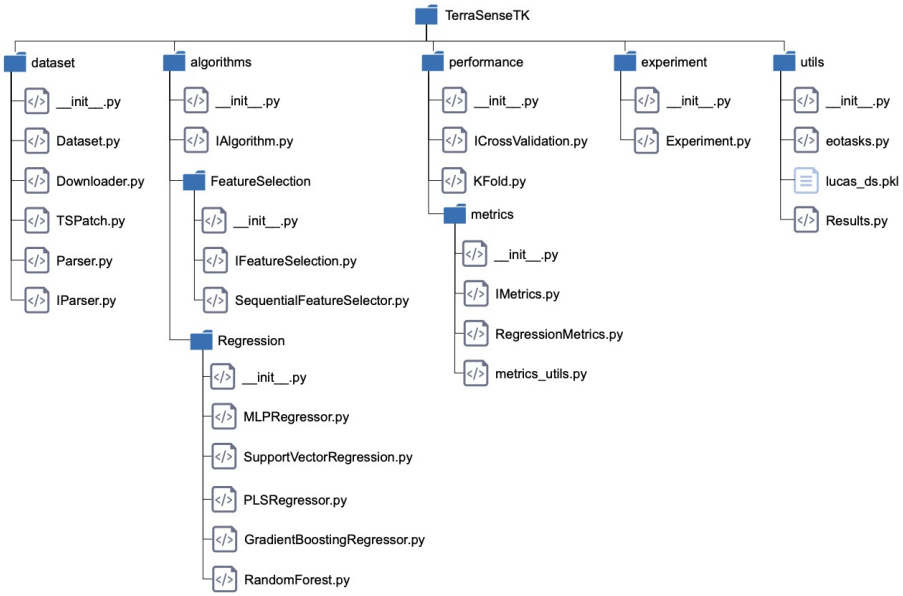


Figure 3.5: TerraSenseTK folder structure [6].

3.3.2 Scikit-learn & Others

sklearn is a popular open-source ML library for Python that provides simple and efficient tools for data analysis and modeling. It is built on Numpy, SciPy, and Matplotlib and is designed to work seamlessly with other scientific and data analysis libraries. sklearn is frequently applied for classification, versatile and well-documented, making it a popular choice for experienced ML practitioners. Thus, it provides a solid foundation for building and experimenting with ML models.

As mentioned before, two other toolkits were also examined to implement SSL models into tstk. They were the Unified Semi-Supervised Learning Benchmark (USB) [41] and LAMDA-SSL (LAMDA) [7]. The first one is represented as a framework that agglomerates 14 state-of-the-art DL SSL classifiers, such as the ones mentioned in [2], [5], and [38]. Moreover, it strikes as an optimal approach of software as the authors compared it with another, now outdated, DL SSL toolkit (TorchSSL), and revealed that it *"largely reduces the training cost"* [41] from 279 GPU days to 39 GPU days. LAMDA however, is an extensive open-sourced Python toolkit made solemnly for SSL, by incorporating both statistical and DL algorithms, both for regression or classification. They offer an interesting amount of 30 unique models, also state-of-the-art and included in USB, plus an additional Data module made for data management and transformation. According to the authors, it *"outperforms USB in various aspects, including the number of algorithms, data types, task types, and functionality"* [7], and conclude the document displaying a table 3.6 with which firmly state that LAMDA is superior to sklearn and USB. To conclude, both these toolkits reveal to be potential sources of optimized software that tstk may use to expand its range of SSL models. However, one must be cautious as USB's interface seems to be outside of tstk's structure, and

Toolkit	scikit-learn	USB	LAMDA-SSL
# deep SSL algorithms	0	14	18
# statistical SSL algorithms	3	0	12
Types of data	Tabular Image Text	Image Text Audio	Tabular Image Text Graph
Types of task	Classification Regression Clustering	Classification	Classification Regression Clustering
Hyper-parameters search	✓	×	✓
GPU acceleration	×	✓	✓
Distributed learning	×	✓	✓
Documentation	✓	✓	✓

Figure 3.6: The comparison of LAMDA with related SSL toolkits [7].

LAMDA is enormously lacking in validation from sources besides its own.

4

Dataset Exploration

Contents

In this chapter, the reader may find a thorough investigation regarding all datasets that were mentioned in Datasets. Each research will be presented throughout 4 smaller chapters: "Introduction & Potential Applications", "Availability & Data Structure", "Data Analysis Methodology", and "Results & Discussion". The first subsection serves as an introduction by lightly describing the dataset's purpose and potential for this research. The second and third components are written to inform the reader about the original data structure, and to justify (and describe) the methods conducted to analyse the data it contains. Finally, the last part is noted as a reflection of the dataset according to its previous analysis, whereupon a decision is made to use it or discard it. All pieces of code executed can be found in the Gitlab repository[1] with a brief description. From this point onwards, all references starting with capital B (`B.#`) relate to this repository.

4.1 Dataset 1: AfSis

4.1.1 Introduction & Potential Applications

As previously mentioned in Section 3.1.1, AfSis is a dataset with 105.074 soil samples with ppm values for nearly all of the sixteen core nutrients described in Soil Fertility, collected from several locations spread through sub-Saharan Africa. Throughout this thesis, we have claimed several times that soil nutrient values are scarce and even though SSL makes use of X_u, it must contain a sufficient amount of X_l to learn the data distribution. So, it seems clear that ignoring such an extensive amount of labeled data is erroneous. Thus, AfSis's potential rises by having a rare quality: substantial amounts of nutrient ppm values.

4.1.2 Availability & Data Structure

The dataset is made available from the AfSis website[2] where each soil property is presented in a GeoTiff file for the whole African continent. Each file contains between 10 to 20 GB of data. Moreover, one can choose between two depth intervals for which the soil properties were generated, being available to download the 0-20 cm depth version, or the 20-50 cm version. Following this, one must make use of a software package developed to work with such heavy files, which can be found on the OpenLandMap website[3], or through Python coding according to the example found on the *eumap* project repository[4]. Both of these options were mentioned by the authors of the paper and thus considered to be trustworthy. Ultimately, all information is extracted to an Numpy array with three dimensions: $heigh \times width \times bands$, which is an optimized data structure, plus can be easily manipulated.

[1] Gitlab repository, https://tinyurl.com/bde5zbhb
[2] AfSis download, http://tinyurl.com/45d4uh4c
[3] OpenLandMap, http://tinyurl.com/5h76nrr9
[4] eumap, http://tinyurl.com/4cdsd6aj

4.1.3 Results & Discussion

The reader may notice that one subsection ("Data Analysis Methodology") was ignored. In fact, due to the nature of this dataset, a decision can already be made, and so no further data investigation is required. Even though it outshines many other possibilities for its large amount of X_l, it demonstrates a lack of reliance on its own prediction by having a R-squared average value of 0.728 [4], and only three of the sixteen nutrients have a R-squared value above 0.8 [4]. Moreover, even when discarding its performance values, the fact that it is composed of synthetically generated data provides little confidence, and to conclude its assessment, it also lacks on the time of nutrient prediction, which must be given to extract EOS data.

4.2 Dataset 2: BEN

4.2.1 Introduction & Potential Applications

Previously described in Section 3.1.2 (Big Earth Net Dataset), BEN is a vast collection of 590.326 image patches from the S2 satellite labeled with land cover classes. Spread through ten European countries, it offers a vast amount of EOS data that can be easily used as X_u. Similar to AfSis, this dataset's value comes with the rarity of encountering such vast source of data, but unlike the previous, it contains real values that were captured between June 2017 and May 2018.

4.2.2 Availability & Data Structure

The data can be downloaded through their website[5] and comes organized in a file hierarchy, having a main directory that contains sub-directories for each image patch. These sub-directories are named after their specifications and strictly follow the pattern: "sentinel-id_MSIL2A_YYYYMMDDTHHMMSS_h-order_v-order", where "sentinel-id" refers to satellite A or B from the constellation S2 (mentioned in Space Missions), "YYYYMMDDTHHMMSS" all together refer to the specific time in 24 hours format of data acquisition, and "h-order" "v-order" the horizontal and vertical *"order of the patch in the tile from which the patch is extracted"* [28]. These sub-directories contain 12 GeoTiff files for each S2 band and 1 more file with meta data. Each GeoTiff file contains a matrix of pixel values for the band it was named after and, with the appropriate software, one can transform it into a Numpy array. The remaining meta data file contains light but important data, where one can find the multiple land cover labels saved in another Numpy array, the id for the original unprocessed tile, the acquisition data, the world coordinates for each corner of the EOS patch, and the projection used to represent those coordinates.

[5] BEN download, https://bigearth.net

Figure 4.1: One randomly selected image patch from each of the four main clusters, plus an additional Cloud patch. From left to right: Vegetation cluster; Snow cluster; Sea cluster; Dry cluster; Cloud patch.

4.2.3 Data Analysis Methodology

Reaching 66GB of data, this dataset is of considerable size and is so computationally heavy that data analyses are costly. Therefore, the following website[6] was used for its investigation. From it, one can discover that each EOS patch corresponds to $1.2 \times 1.2km$, and most are classified with several (sometimes reaching 6) labels. Moreover, one can form groups of visually homogeneous image patches and count the number of subdivisions in the dataset, plus discover which subgroups contain the most data by applying a similarity clustering focused on each image patch's features and ignoring their land cover classes. This approach revealed that BEN is mainly composed of four groups, which can be summarized as 375.565 (63.62%) patches with vegetation, 67.982 (11.52%) filled with cloud coverage or snow, 56.842 (9.63%) patches of the sea, and finally 18.765 (3.18%) of dry arable land. Examples from randomly selected images are provided in Figure 4.1.

4.2.4 Results & Discussion

As our research is focused on studying the advantages that SSL has on SL when predicting soil nutrients, it is trivial to decide which clusters should be maintained (any image with soil). Therefore, all sea, snow, and cloud image patches are removed from this dataset, totaling around 394.330 data points that can be used as X_u. However, one must take into consideration the dimensions each EOS patch has $(1.2 \times 1.2km)$, which covers an area of $1.44km^2$. Such information is crucial since it is unrealistic to assume that all that area contains the same amount of nutrient values, and so, to be used, one must split each image patch into a $50 \times 50m$ square. Unfortunately, BEN only associates each land cover class to the full image patch (justifying the number of multi-labels one image patch can have), becoming impossible to identify through BEN what land cover class must be associated with the filtered $50 \times 50m$ square. It is true that one could obtain several $50 \times 50m$ squares from the same image patch, and then calculate to which coordinates its corners are mapped, and extract the correct land cover label from the CORINE dataset. Nevertheless, this approach strikes as a steep deviation from the investigation's natural course, and so it is set aside. Thus, one can conclude that even though BEN seems highly

[6] BEN visualization, http://tinyurl.com/3vc7pa36

promising and can indeed be utilized, it should be temporarily discarded to focus on the main task.

4.3 Dataset 3: NCSS

4.3.1 Introduction & Potential Applications

Following the earlier delineation written in Section 3.1.3, NCSS is an enduring and highly informative dataset, as it has been ongoing since 1899 and maintains numerous soil properties for each of its locations. It has a huge potential to be used as X_l for the enormous quantity of accurate soil data that the authors mention to contain: *"500,000 hectares have been identified and mapped in the United States"* [30], and as stated before, it is a hard characteristic to come across. Therefore, the same principle from the prior explored datasets applies to NCSS: it must be considered for being a rare source of detailed soil data.

4.3.2 Availability & Data Structure

The dataset can be download from the official NCSS website[7] in SQLite format or Microsoft database format. It was decided to follow the first option for having a straight-forward interface from the "sqlite3" terminal command. The data comes structured in a relational database, which represents datasets through the combination of many interconnected tables, splitting data into different categories, such as the "lab_chemical_properties", which contains all nutrient values, or the "lab_physical_properties", that includes bulk density. They maintain connections to other tables through value matching in pre-defined columns. This dataset is made of 20 unique tables and 1.004 features, some to identify/map each data point, but most related to soil properties. It's structure may be visualized in Figure A.1.

4.3.3 Data Analysis Methodology

Even though relational data tables are an efficient strategy to save and organize data, it lacks on straight-forward methods to examine its content. Allied to the fact that Pandas is an extremely optimized Python toolkit to explore data, before its exploration, this dataset was compacted into a file format understandable to it: Comma-Separated Values (CSV). Contrary to a relational database, these files do not split the dataset's content into unique tables, but represent them in a single file, where each line corresponds to a data instance and are filled with comma-separated values per feature. Such structure makes them simple and easy to use.

[7] NCSS download, http://tinyurl.com/ypd3zbxs,

4.3.3.A Conversion from SQLite to CSV

Recalling our main objective, our final dataset must contain EOS data as features, and nutrient values as targets, and since the NCSS dataset solemnly has soil data, it is perfectly suitable to be used as Y. Thus, it is intended to create a singular CSV file where each line contains some sort of identification, date and location of nutrient extraction to associate with EOS data, and naturally, the nutrient values. To do so, one must unite all tables that possess the prior mentioned necessary information as their columns, and ignore the remaining ones. After this unification, all features that come with the selected tables that are not seemed as relevant are removed. Four of the sixteen nutrients referred in section Soil Fertility are missing, being the remaining twelve C, N, P, K, Ca, Mg, S, Cu, Fe, Mn, Mo and Zn. Moreover, these nutrients are spread through three different tables (lab_chemical_properties; lab_physical_properties; lab_major_and_trace_elements_and_oxides) and present heterogeneous unities of measurement. Some are represented in $gravimetric\%$, others in mg/kg and the remaining in $meq/100g$, which provokes erroneous predictions. Therefore, bearing in mind that most investigated soil fertility papers make use of ppm, the entire nutrients are converted to it. These conversions are easy to calculate and go according to the following formulas: $ppm = gravimetric\% \times 10^4$, $ppm = mg/kg$, and $ppm = meq/100g \times M \times 10$, being M the molar mass of the atom in question. Regarding the identification features, the date of extraction can be found in table "lab_pedon" column "observation_date", and the longitude and latitude in table "lab_site" spread through columns "longitude_std_decimal_degrees" and "latitude_std_decimal_degrees", respectively.

Having explained NCSS's structure and its data types, the reader can effortlessly comprehend the dataset's conversion from a relational database in SQLite to a single CSV file. It was performed with the "sqlite3" command through the machine's terminal and strictly followed the mappings displayed in Fig. A.1. The conversion process can be visualized in the following diagram Fig. A.2 Firstly, a unification of the latitude and longitude values with an internal table called "lab_layer" was created since the second has a direct connection to all tables with soil properties. So, a table called "ids_loc" was built from the merging of "lab_site" with "lab_layer" through the column "site_key" (code displayed in `B.1`) resulting in 417.652 data points. Following this, each of the three tables that contain nutrients were sequentially merged with "ids_loc" through columns "labsampnum" and "layer_key", creating a table with 32.586 rows `B.2`, which was then merged with table "lab_pedon" through keys "pedon_key" and "site_key" `B.3` to append the date of nutrient extraction. The final table consisted of 32.583 data rows. Ultimately, the only step missing is the removal of irrelevant columns that came with these unifications, which was managed by writing some more code: `B.4`. Hence, we reach a singe table, named "data", with which we can easily convert to a single CSV file.

4.3.3.B Data Analysis

From this point, we could begin to utilize the Pandas toolkit to deeply explore the dataset. The first investigation was focused on discovering the missing data to discard the data points that lack on crucial features. Any row with missing date, latitude, or longitude was removed since they are impossible to associate with EOS data. From this parsing, we end up with 23.489 data rows. The code for this step can be found here `B.5`

Another feature that could invalidate data when absent is the depth of extraction, as EOS focuses on the upper layer of soil. Therefore, for this investigation, only data points close to 30 cm depth are considered to be superficial. In this dataset, depth is represented as an interval obtained from two columns, ”hzn_bot” and ”hzn_top”, where ”bot” is the deepest depth and top the upper depth limit. The same location has several analyses, each per depth interval. Thus, any row with both of these columns missing is removed, plus all data that has less then 50% of its depth interval below the 30 cm limit will also be discarded. Furthermore, since some locations have multiple rows that satisfy this limit and our ML models cannot have the same EOS pointing to different Y values, we must keep only one soil analysis per location, being the most precise the one closest to the surface. This procedure is obtained with code `B.6` and culminated in 5.101 data points.

There are many ways to measure a nutrient’s value, such as the Bray-1 and Mehlich-3 methods, and to increase accuracy, NCSS dataset conducted several techniques to reach the same objective. I.e., some nutrients were analyzed with multiple procedures, resulting in tables with multiple columns for the same nutrient. For example, there are P values in columns ”phosphorus_bray1”, ”phosphorus_bray2”, ”phosphorus_major_element” and others. As they are correlated and have the same purpose, one could merge them into a single feature that is able to represent them all, and calculating their average strikes is a natural approach to do so. However, as mentioned before, these features are measured in three different unities and must have the same representation (ppm) to be correctly merged. This procedure can be viewed in `B.7`. Subsequent to their convergence, we investigated the number of missing values per nutrient feature. From executing this Python code in `B.8` and after evaluating the table it creates, it was unexpectedly discovered that a lot of the data is absent, as some nutrients have more than 70% missing data 4.1. From this discovery, one must verify if there are data rows where all nutrients are missing, and if so, remove them from the dataset. Code `B.9` was executed to achieve this and resulted in discarding seven data points, leaving 5.096 rows with partial nutrient values.

As the second to last step for NCSS’s research, we dove into the date of extraction feature, with the natural explanation that data points refer to times prior to the oldest satellite, from which we are capable of acquiring EOS data, are irrelevant. From this parsing, we obtained 5.083 ready-to-use data points. It was also counted how many data rows are in accordance with each of the satellite’s life span `B.10`, from which we got 3.680 included in LMSS, 4.102 in LTM, 1.078 in LETM, 177 in LOLI and 162 in S2.

To close this dataset's analysis, one must also identify how many of these data points can be associated with EOS data. Ultimately, a SSL model must be fed with X_l or X_u, and both contain features. Thus, this dataset's viability becomes dependent on the amount of data points, from the final 5.083, that can be associated to band data. To do so, after having split the data into different CSV files, each with data inside the satellite's life span, one must execute a Python file that utilizes tstk and extract all EOS data for each point. However, not all satellites seem as an adequate option, some for their characteristics, and others for the size of data that is available. Notice that only 177 and 162 soil analysis are in LOLI and S2 life span, and so, even if these satellites had captured data from all those locations at their date of extraction, they would be too small to create an accurate ML model. Moreover, even though LMSS has an interesting amount of data, it not only has low resolution sensors (60m), but also has a short variety of bands (G, R, Ultra-Red and NIR). Therefore, LMSS data will also be extracted, but only used as a last-resource. So, LMSS, LTM and LETM were used to extract EOS data and associate them to their respective CSV files `B.11`, from which LMSS extracted 1.107 EOS patches, LTM got 1.018 data correspondences, and LETM 267.

Nutrient	Missing count	Missing percentage (%)
C	3057	59.93
N	1301	25.50
P	3335	65.38
K	24	0.47
Ca	27	0.53
Mg	24	0.47
S	3719	72.91
Cu	3928	77
Fe	2567	50.32
Mn	1645	32.25
Mo	4281	83.92
Zn	4026	78.93

Table 4.1: Missing values count and percentage per nutrient in NCSS

4.3.4 Results & Discussion

The first immediate conclusion that we can take is the enormous reduction of data. NCSS started as a promising large dataset full of nutrient values, perfectly suitable to be X_l, and even though all filtering procedures were reasonably justified, it still went from 417.652 data points, to 32.583, and eventually just 5.083. Such reduction, around 98.78% to be exact, diminishes the dataset's potential and more so its confidence, when mentioning the incomprehensible large amount of missing value percentages referenced before 4.1. Secondly, not all 5.083 soil locations are usable for X_l, as only a small percentage can be associated to EOS. In fact, true X_l potential only comes from the 1.107 LMSS data patches, or the 1.018 extracted by LTM, being the other two too small for ML creation. To overcome

this low dimension matter, one could combine the datasets into a singular file, where the model would not know the bands' sources, which is possible to do so when the satellites share similar sensors. LTM, LETM and LOLI possess the same resolutions (30 m), but LMSS is limited by 60 m and S2 with 20 m. This incompatibility comes from the necessity to apply pixel transformation to reduce or increase the resolution into 30 m, which could compromise the model's accuracy in exchange for 1.269 (at most) data points. Therefore, one can merge LTM with LETM and LOLI adding up to no more than 606 EOS patches. So, this dataset has proven to be quite a surprise for only 1.22% of its content are relevant soil locations, and from it, just around 1.018 are available to be X_l.

4.4 Dataset 4: LUCAS

4.4.1 Introduction & Potential Applications

Last but not least, we investigate the LUCAS datasets. Referenced in LUCAS Topsoil & LUCAS Copernicus, one possesses precise soil property values carefully measured in laboratories, and the other a strictly followed land cover and usage class identified in a delimited area. Naturally, an accurate model that predicts nutrient values must be trained with data that holds such information. Here lies LUCAS Topsoil's potential: it contains rare and laboratory analysed nutrient data. Contrarily, LUCAS Copernicus lacks on such details, but it also has value, since it's data contains a high detailed land cover/usage label that strictly follows a classification rule, which has been proven to increase performance when filtered [6]. Moreover, it associates these properties to an area, which smoothens the EOS data extraction and increases precision. Bear in mind that all other datasets solemnly presented locations as geographical points, and since satellite patches have resolutions much higher than a single point in space, our band data would become a single pixel. Thus, having an area of investigation per data entity highly increases the investigation's confidence. Allied to this, being from the same organization, conducted during the same time periods, and mostly executed on the same sites, they can be merged to form a LUCAS dataset that maps land cover/usage data to nutrient values, and fits perfectly as X_l. Moreover, as mentioned before, these datasets purposely collect data from locations where S2 satellites capture image patches, and so one can hope that most of their data can be mapped to EOS data (unlike NCSS). So, having a considerable amount of 18.984 soil nutrients, an additional land cover and usage class, association with EOS data and an area delimitation, these datasets strike with a strong probability to become X_l.

4.4.2 Availability & Data Structure

Both datasets are available through the ESDAC website and come in CSV file format, being the LUCAS Topsoil available here [8] and LUCAS Copernicus here [9]. Like all other CSV files, these datasets are made of several rows of data, each with one value per feature separated by commas. From a variety of available features, our interest lies on any that have nutrient quantities, spacial location, time of extraction, and land cover class. Moreover, it was decided to only maintain nutrients N, P and K, since they are the most important minerals 2.1. Therefore, this research discards all features that come from these datasets with exception to LUCAS Topsoil "Phosphorus", "Total nitrogen" and "Extractable potassium" columns, and "area", "LC0_Desc", "LC1_Desc" and "LC2_Desc" from LUCAS Copernicus. Features with identification values were also kept to merge both datasets, them being "POINT_ID", "TH_LAT" and "TH_LONG" columns.

All nutrient features are convertible to ppm, being N the only that requires computation. P and K features are in mg/kg, and N in g/kg. According to [32], some nutrient values are absent and others presented as string "¡ LOD", which reference missing laboratory analysis and outside of limits of detection, respectively. The "area" field, which is only available in data points from LUCAS Copernicus, is represented as a Polygon object, which comes from a Python toolkit named Shapely built to represent geometric figures and can be used with world coordinates. "POINT_ID" is a number that identifies each analysis, and "TH_LAT" "TH_LONG" columns when together pin point the exact location in geographical coordinates WSG84. Lastly, land cover features come in string values dispersed through a discrete space. These features are named with a number that represents the level of classification detail, being 0 the most general, and 2 the most specific. "LC0_Desc" has 8 unique values (Artificial land, Cropland, Woodland, Shrubland, Grassland, Bareland, Water and Wetlands), each with a set of unique "LC1_Desc" labels, which also contain another sub-level of land cover classes specified with "LC2_Desc". Take for example label "Common wheat" from "LC2_Desc". It represents all $Triticum\ spelta$ plants, its upper class is the "LC1_Desc" with value "Cereals" that descends from "LC0_Desc" feature value equal to "Cropland".

4.4.3 Data Analysis Methodology

For the reasons we mentioned before, both LUCAS Topsoil 2018 and LUCAS Copernicus have potential and even more when merged together. Therefore, the datasets were combined through their "POINT_ID" column and formed a new dataset called LUCAS. It has all the relevant features mentioned above spread through 18.984 data points, all with land cover labels, but some lacking the feature "area".

The first step was to associate EOS data to each point of the dataset. Being specifically built with

[8] LUCAS Topsoil, http://tinyurl.com/cndpyte6
[9] LUCAS Copernicus, http://tinyurl.com/4mt38my8

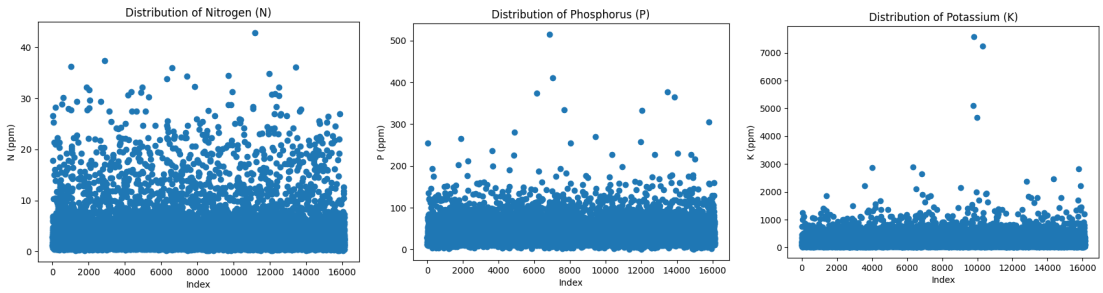


Figure 4.2: N, P and K distribution in LUCAS dataset.

soil data from locations frequently passed by S2 satellites, it was decided to only obtain data from it's collection. Moreover, to increase the chance of correspondences, all data 10 days prior or 10 days posterior to soil extraction were considered. Regarding the resolution, as it was displayed in S2 band information., S2 important sensors possess resolutions of 20 m, and to avoid any transformations, 20 m were also specified to extract band values. All EOS data that has more than 10% cloud coverage was discarded. This process was conducted with Python coding that makes use of tstk toolkit, which can be viewed in the following appendix `B.12`. From it, we obtained 16.130 correspondences.

Subsequently, we dove into the nutrient data to count the amount of values that might be absent, plus to identify how they are distributed. After running a simple line of code `B.13` we verified that both N and K are mostly present (12 and 34 missing values), whereas P presents a considerably larger amount of 4.293 absent data. Regarding their distribution, another Python toolkit named Matplotlib[10] was utilized to draw a scatter plot for each nutrient. From the following lines of code `B.14` we obtain these three images Fig. 4.2. Through their analysis, one can perceive that all three nutrients are stable, as most N dots are below 10 ppm, P values beneath 100 ppm, and almost all K dots are lesser than 1000 ppm. In fact, according to the IQR method, roughly 8.48%, 4.81% and 5.31% of N, P and K values are outliers. These calculations considered the total amount of nutrients to be the number of data rows minus the missing and "¡ LOD" values. These percentages were obtained by visualizing the following table Table 4.2 which was created from this code `B.18`. This data serves also as a confidence gain since no dots are below 0 ppm. Lastly, when investigating the land cover classes, one discovers that the dataset is considerably unbalanced. As we can perceive from the following histogram Fig. A.3, obtained by executing this piece of code `B.16`, 6.427 data points are in "LC0_Desc" "Cropland" class, 5.120 in "Woodland", 3.294 in "Grassland", and the remaining 1.289 dispersed through 5 other classes. Despite this, our focus is on soil labels and crop identified lands, which we have 2.281 of "Grassland without tree/shrub cover", 1.745 of "Broadleaved woodland" and 1.350 of "Common wheat", as one can visualize from Fig. A.4 which was created with `B.17`.

[10] Matplotlib, https://matplotlib.org/stable/

Nutrient	Count	Mv	Mean	Std	Min	25%	50%	75%	Max	Outliers
N	16.021	12	3.038	3.561	0.2	1.3	1.9	3.3	42.8	1.359
P	11.740	4.293	34.671	27.481	0.3	16.5	26.4	43.9	515	565
K	15.999	34	206.155	213.253	8	84.6	153.5	263.75	7578.8	850

Table 4.2: LUCAS N P K summary. Mv stands for missing values

4.4.4 Results & Discussion

Recalling the scatter plots displayed above Fig. 4.2 and the number of outliers presented in the previously shown Table 4.2, one can firmly state that LUCAS dataset is stable, where most of its data is compacted in an interval of values. Moreover, solemnly 2.846 data points were unsuccessfully associated with EOS data, leaving an interesting amount of 16.130 data points ready to build the ML models. However, even though N and K data are ultimately present, it is true that many P values are absent, but since our ML models are SSL and utilize unlabeled data, such absence does not raise an alarm. In fact, having EOS data from unlabeled points that have a land cover class serves greater than extracting band data from randomly selected locations, which are not guaranteed to be in the intended domain. I.e. with this set of X_u one can still split unlabeled data into land cover, whereas a random selection could bring unrelated soil. Thus, this dataset strikes as an improved version of NCSS. Furthermore, it presents a satisfactory amount of data, and when discarding all lands that are unrelated to soils, them being classified with "LC0_Desc" values "Artificial land", "Wetlands" and "Water", it still outputs 16.033 data points. Unfortunately, it offers a small amount of data when grouping by the "LC1_Desc" feature, but it can be enough to increase the model's performance. Therefore, even if it is a small amount of data, this dataset offers the possibility to test the conclusions made by [6] that models preform better when trained on crop filtered land and more importantly, to evaluate the SSL expected gain in these conditions.

To conclude, this dataset has proven to be stable, possess little but relevant data, contains crucial land cover information that could increase performance, and so serves as a contribution to our cause.

4.5 From Regression to Classification

In this extra section, the reader may find the justification for our decision to approach this research as a classification problem, in spite of the natural tendency it has for regression. This proposal rose from an interesting book [8] which, in chapter *"Soil: Fertility and Nutrient management"*, revealed that crop yield is only dependent of an interval of concentration values, and not a precise amount. The author stated that, when inside certain concentration values, *"increasing nutrient supply has no effects on growth"* [8]. From this observation, one can split the otherwise infinite nutrient concentration domain, to a discrete space of three values. One that represents all nutrient concentrations that give plant growth, other for any amount that produces deficiency side effects, and another that creates toxicity. This approach was

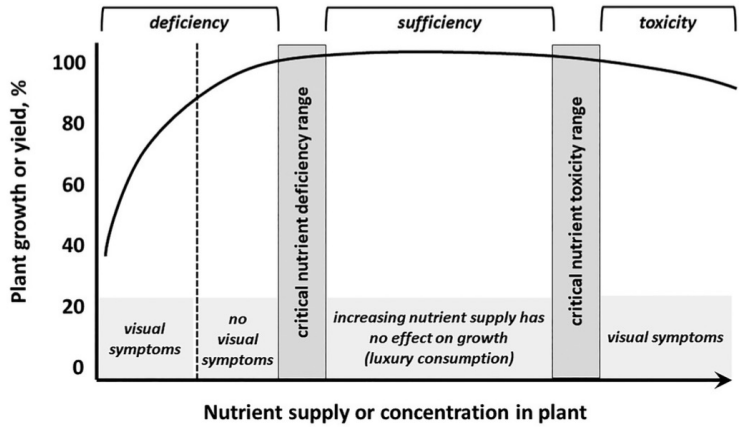


Figure 4.3: *"Relationship between plant nutrient concentration and plant growth. The critical nutrient deficiency range represents the nutrient concentration below which nutrients should be added; however, nutrients added beyond this level increases plant nutrient concentration without a response in plant growth or yield."* [8]

made by the author and can be better understand in the following graph Fig. 4.3.

So, being proven that there is no need to know the precise concentration of nutrients, but instead boundary values that separate these categories (deficiency, sufficiency and toxicity), giving importance to treat this research as a classification problem, one must investigate how it can be transformed from regression. The first decision is the number of classes. It is true that the author clearly differentiated the deficiency interval from the toxicity one, as they have divergent consequences. However, our purpose is to solemnly identify if a soil is fertile given its EOS data, not its state of infertility. Therefore, we can classify our soils with two labels: fertile and infertile, where infertility incorporates the prior deficiency and toxicity state. Regarding the core conversion process, it is certain that there are more than one approaches to conduct it, nonetheless it was decided that a domain-driven method would be more realistic and so, precise. Therefore, it is intended to discover an interval of values representing all nutrient concentrations that maintain plant growth. To do so, several papers that investigate the relation between concentration values and crop yield were read, which clearly state that the precise limit is highly dependent on the species, and sometimes even subspecies, of the plant being grown. I.e soil fertility is described as a concept interconnected to the plant in question. One cannot say that a soil is fertile without knowing what plants it grows, for some species it is lacking in nutrients, and for others stimulates growth. One of those papers mentioned the heterogeneous amounts of critical P values in maize, wheat and rice: *"The average critical Olsen-P values (...) were 18, 14 and 11 mg kg−1, respectively."* [42], whereas another displayed the divergence in N limits for several crops, which are shown in Fig. 4.4.

Summarizing four different papers ([43] [44] [45] [46]), one can state that the general critical concentration values for N must be between 10 and $50\ mg/kg$, P *"optimal crop yield ranged from* $10,9\ mg/kg$ *to* $21,4\ mg/kg$*"* [42], and K must be around 40 and $80\ ppm$ [47]. Regarding $Triticum\ spelta$ crop, this

	Diagnostic Range (% Dry Mass of Leaves)		
Type of Crop	**Low**	**Sufficiency[a]**	**High**
Agronomic Crops			
Grass grains	<1.5	1.8 to 3.6	>3.6
Legume grains	<3.6	3.8 to 5.0	>5.0
Cotton	<3.0	3.0 to 4.5	>5.0
Tobacco		4.1 to 5.7	>5.7
Rapeseed		2.0 to 4.5	>4.5
Sugarbeet		4.3 to 5.0	>5.0
Sugarcane	<1 to 1.5	1.5 to 2.7	>2.7
Bedding Plants		2.8 to 5.6	
Trees			
Conifers	<1.0	1.0 to 2.3	>3.0
Broadleaf	<1.7	1.9 to 2.6	>3.0
Cut Flowers	<3.0	3.1 to 4.7	>5
Ferns		1.8 to 2.9	
Potted Floral		2.5 to 4.2	
Forage Crops			
Grasses	<1.5	2.0 to 3.2	>3.6
Legumes	<3.8	3.8 to 4.5	5 to 7
Tree Fruits and Nuts			
Nuts	<1.7	2.0 to 2.9	>3.9

Figure 4.4: *"Concentrations of Nitrogen in Leaves of Various Crops under Cultivated Conditions. Note: (...) Low is value where symptoms of deficiency are showing. Sufficiency is mean range of lower and upper concentrations commonly reported in healthy plants showing no deficiencies. High is a concentration that might represent excessive accumulation of nitrogen."* [9]

paper [48] states that soil N concentrations should be between 16.300 and $30.100\ mg/kg$, P in 2.650 to $5.020\ mg/kg$, and K roughly 3.060 to $5.600\ mg/kg$. These values are inconsistent with the previous general values, and furthermore, only P is slightly coherent with Table 2.1. In fact, as one can perceive from Table 4.2, no nutrient distribution goes according to these values. From these observations, it becomes clear that the dataset should be split into crops and then converted given the specie's critical nutrient values. Despite this, moving forward with said approach would mostly reduce our dataset from 16.130 to 1.350 data points, which limits the investigation. Therefore, to avoid discarding such quantities of data, two conversions were proposed. One utilized general critical points for soil fertility and included the whole dataset ignoring the land cover labels, whereas the other solemnly possesses $Triticum\ spelta$ plants and transformed its nutrient values to fertility classes according to this species behaviour. Thus, to preform these two methods, a deeper investigation had to be conducted to discover values for general and $Triticum\ spelta$ concentration boundaries. It is true that such discrepancy is hard to grasp, however, and as previously mentioned in Section 2.1, soil fertility is greatly influenced by many properties (most are not included in this paper to maintain a clear research flow, and can go from the soil's exposition to light, to its inclination, and bulk density), which hardens a general classification process that only takes into account land cover. So, allied to the fact that there is few research investigations that could provide us with certain values, this second dataset approach also utilized the general concentration values of 10 to $50\ mg/kg$ for N, $10,9$ to $21,4\ mg/kg$ for P and 40 to $80\ ppm$ for K.

4.6 Final Dataset

Having explained the classification process, it is possible to conclude this chapter with an overview of a final dataset. Built regarding all prior information, it simply consists of the remaining 16.033 data points from LUCAS resultant of removing any row with ”LC0_Desc” feature equal to ”Artificial land”, ”Wetland” or ”Water” from the original 16.130, plus having applied the conversion to classification with the following code in `B.19`. This approach fails to make use of the NCSS data since the only relevant information it brings is attached to EOS band values from satellite LTM, which resolution is incompatible with the S2 satellites used for LUCAS when discarding image transformation as a possibility. Additionally, using NCSS by itself seems to be an irrelevant approach for the small amount of data it provides (1.018). Thus, for the remaining of the document, the word ”dataset”, or any alike, is ultimately referring to the manipulated LUCAS agglomeration of data.

Nutrient	Count	infertile	infertile%	fertile	fertile%
N	16.021	15.302	95,51	719	4,49
P	11.740	7.772	66,20	3.968	33,78
K	15.999	13.513	84,46	2.486	15,54

Table 4.3: Final dataset's N P K balance according to general classification.

Nutrient	Count	defice	defice%	fertile	fertile%	toxic	toxic%
N	16.021	15.302	95,51	719	4,49	0	0
P	11.740	574	4,89	3.968	33,78	7.198	61,31
K	15.999	1.078	7,36	2.486	15,54	12.335	77,1

Table 4.4: Final dataset's N P K balance according to general trinary classification.

Having constructed a terminal dataset, and bearing in mind the transformation to classification, another set of evaluations were performed to better visualize how its content is presented. Firstly, a balance assessment was made through the following piece of code in `B.20`, from which we obtained Table 4.3, that reveals the enormous unbalance both N and K possess. Despite this, all these nutrients were plotted (Fig. A.5) to better analyze their distributions, and when colored according to their inclusion or exclusion from the general fertility limits, one can identify how little the fertile amounts are represented. To draw these plots the code found in `B.21` was executed. Moreover, one can observe that most N fertile values are in a lower density zone, which raises doubts towards the N critical concentration values certainty and the dataset's realness. However, since there is no reason to believe that LUCAS topsoil data is unrealistic, perhaps it is more frequent to find infertile values, specially N values, than concentrations that give crop yield. P and K might become more distributed if dispersed through the three original classes. Therefore, it was investigated how they are scattered when all values below the general critical concentrations are labeled as ”defice”, those inside the limitations classified as ”sufficient” and the remaining ones as ”toxic”. After executing the following code in `B.22` we can view the plots shown in Fig. A.6 from

which we can conclude that most data is considered to be toxic, and so invalidating trinary classification as a solution to balance the dataset. This Table 4.4 presents the concrete values per class, which is obtainable from running the code written in B.24

Metric	Value	Percentage
Count	16.022	100
insufficient	0	0,0
low	15.311	95,56
caution	541	3,38
good	170	1,06
high	0	0,0

Table 4.5: Final dataset's N balance according to extended classification.

Regarding N balance values, another solution found involved the distribution of N through 5 different classes, but as seen from table 4.5, it failed to overcome this problem.

Nutrient	Count	infertile	infertile%	fertile	fertile%
N	1.350	1.347	99,78	3	0,22
P	1.218	907	74,47	311	25,53
K	1.350	1.282	94,96	68	5,04

Table 4.6: Final dataset's composed only of wheat N P K balance according to general classification.

Nutrient	Count	defice	defice%	fertile	fertile%	toxic	toxic%
N	1.350	1.347	99,78	3	0,22	0	0
P	1.218	24	1,97	311	25,53	883	72,5
K	1.350	2	0,15	68	5,04	1.280	94,81

Table 4.7: Final dataset's composed only of wheat N P K balance according to general trinary classification.

Moreover, even when only considering the dataset composed of only wheat, it still managed to present its imbalance, as one can perceive through the two tables Table 4.6 and Table 4.7. Thus, this observation undoubtedly shows the enormous unbalance that the dataset possesses, and bearing in mind the harmful impact unrepresented data has on classifiers' performances, we are forced to remove N and K from this research.

5

TerraSenseTK: Toolkit Improvements

Contents

In this chapter, the reader may find a clear and detailed explanation of all modifications applied to the tstk toolkit. Divided into seven sections/subsections for each major alteration, one is confronted with a brief description of the previous toolkit's state and the problems it arose, followed by a logical solution to overcome them. All written with a strong reference to two graphs that portrait the prior and posterior state of the modification.

5.1 Introduction

As mentioned before in Section 3.3.1, tstk is a python toolkit built to facilitate ML computation over EOS data. It combines all major steps required to conduct these types of experiments, by gathering all band data for each pre-defined location of interest, encapsulating them in a single optimized `Dataset` object, feeding it to a ML model of choice, and then computing evaluations.

Despite its previous presentation, one must be aware of two concepts used in the tstk implementation to fully grasp the upcoming modifications. They are `EOPatch`[1] and `EOTask`[2] python objects, both developed by Sentinel-Hub package `eolearn`. Capable of containing several variables up to 4 dimensions, `EOPatch` objects are data containers *"designed to store all types of EO data for a single geographical location"* [49]. `EOTask` however, exist to manipulate, add and remove data within an `EOPatch` or even create/delete `EOPatch`. As displayed in Figure 5.1[3], `EOPatch` can have multiple features varying in dimensions. One temporal dimension, two to specify size (height, width), and one dimension for the number of channels (or bands). So, practically one `EOPatch` can have the temporal data of all available bands from one geographical zone.

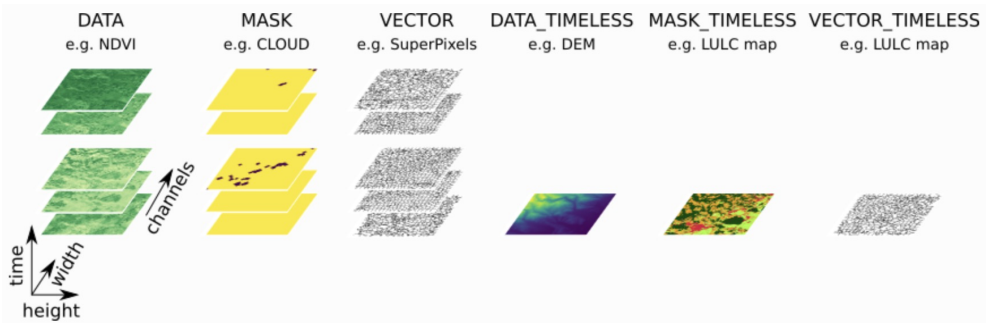


Figure 5.1: Six of the eleven types of features `EOPatch` can represent simultaneously. Notice how they differ on the number of dimensions

The original structure of tstk is presented in Figure 5.2, showing the four key modules: 1) dataset, 2) algorithms, 3) performance, and 4) experiment, and 5) results. Each component was updated in this

[1] `EOPatch`, http://tinyurl.com/4d934j3y
[2] `EOTask`, https://tinyurl.com/yck7mfre
[3] `eolearn.core` overview, http://tinyurl.com/4d934j3y

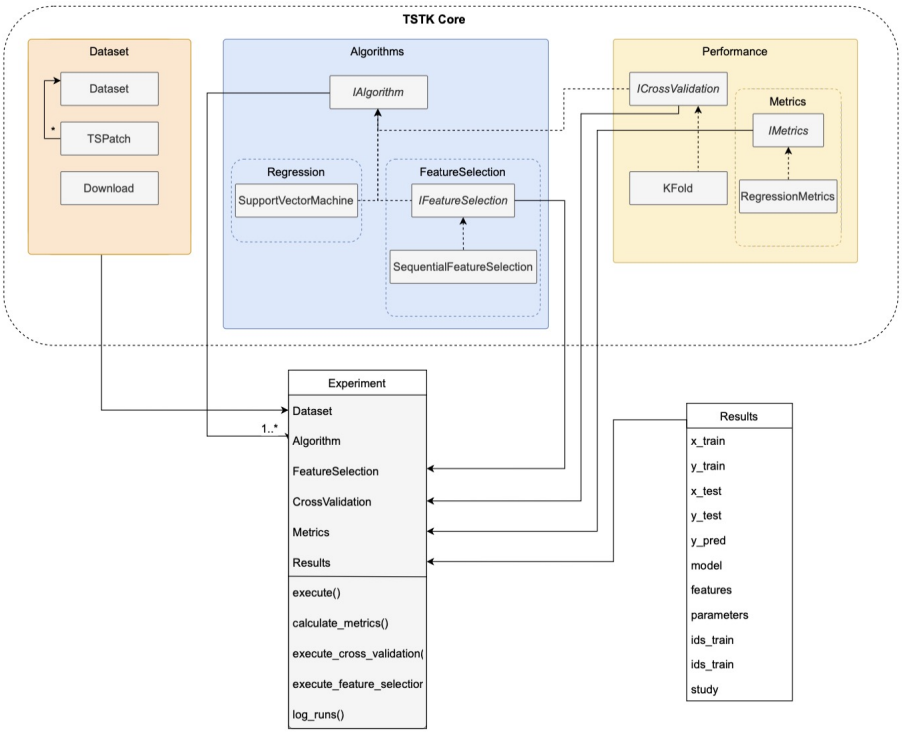


Figure 5.2: Component diagram of the original version of TerraSenseTK [6].

new version, and the modifications can be found online [4].

In the remaining sections of this chapter, we present the most significant modifications that were performed to the original version of the toolkit. Similar to the code references made in the previous chapter, the following sections also references the same Gitlab repository[5] with a `B.#`.

5.2 Module: Dataset

5.2.1 Class: Downloader

The `Downloader` class was implemented to extract EOS data from the Sentinel-Hub API[6]. Totaling seven attributes and seven methods, this class is meant for the user to initialize it with a file containing all the identifications, locations, and nutrient data. Following this, the user had to call `get_bbox()` to compute an area for each location in the dataset in the form of a `sentinelhub.BBox`, so that one could finally execute `download_images()` to extract band data for each bounding box.

This implementation revealed several problems that had to be taken care of. The most aggravated is the lack of generalization and resource management since it only had access to S2 data collection

[4] tstk code, https://tinyurl.com/2kr8djb8
[5] Gitlab repository, https://tinyurl.com/bde5zbhb
[6] Sentinel-Hub, https://www.sentinel-hub.com

and assumed information only present on LUCAS. Furthermore, some attributes could be merged to save memory and other functions could be removed to avoid more computation load. Thus, a complete restructure was conducted.

5.2.1.A Modification: Class inheritance

The first significant change focused on surpassing its lack of generalization. To overcome this problem, the `Downloader` class was converted to an abstract class, and five other classes were created to inherit from it. These new classes represented the four previously mentioned Landsat satellites, plus the S2 sentinel. Written in few lines, these sub-classes only specified the `downloader`'s attributes where these satellites diverged, such as their name, the Sentinel-Hub's variable for their collection, and their available bands. This new structure can be visualized in Figure A.7.

5.2.1.B Modification: Attributes

Secondly, we focused on the class' attributes. Firstly, the five different `GeoDataFrame` variables were merged to a single one (`dataset`), as they were used to represent the same information. Attribute `config` was added to save the Sentinel-Hub's account, which gave errors when absent, and `groundtruth_col` to identify the column of the dataset with the locations' area. `keep` was introduced to to identify all columns in the dataset intended to be saved with the EOS patches extracted from the `download()` method, and finally, `id` was implemented to specify the dataset's column that identifies each data row and consequently names all EOS patches.

5.2.1.C Modification: Area association

The first transformation concentrated on area association to each EOS data. Before our changes, the user had to run `get_bbox_with_data()` method, which calls `get_bbox()` to append a geometry to the `_dataset_bbox` GeoDataFrame. Both these methods seemed overwhelming and confusing, so a solution was derived. Instead of forcing its call, the process was moved to the class' constructor by simply using a special function designed to read strings that represent polygons (`shapely.loads()`), which created a `shapely.Polygon` from reading the `groundtruth_col` and associated it to the `dataset` attribute. With this implementation, tstk conserves computer resources by not requiring a `CostumGridSplitter` and an additional three other GeoDataFrame. The old implementation can be viewed in `B.25`, and the new implementation in `B.26`.

5.2.1.D Modification: Area calculation

The previous alteration led to all data rows with missing `groundtruth_col` values having their areas represented as a single geographical point, from which Sentinel-Hub cannot extract EOS data. Since tstk must also be able to obtain band values from locations with absent `groundtruth_col` values, a new method was created `calculate_area()` which transforms all geometries in `dataset` from a `shapely.Point()` to a `shapely.Polygon()` object that represents a square with the center in the `shapely.Point()` and area specified by the user. This method is made possible by converting our coordinates in WGS84 projection to UTM, and making use of `shapely.buffer()` method that performs these calculations. The code for its implementation can be viewed in `B.28`, which uses a function from the `utils.util.py` file written in `B.27`. Besides reducing overhead, this implementation extends tstk's utility as one can create `shapely.Polygon` with any area value centered in their data locations, extracting unlimited dimensions of EOS data.

5.2.1.E Modification: Geographical visualization

An additional function was created for the user to geographically locate and visualize the polygons that will be used to extract band data from the Sentinel-Hub API. This new implementation was a simple but extremely useful and relevant extension, allowing users to pinpoint the area of data extraction with precision. This novelty was built according to `B.29`, and it displays images like the ones portrayed in Fig. A.8.

5.2.1.F Modification: Downloader method

The `download_images()` method is responsible for associating EOS data extracted from the Sentinel-Hub API to each location in the dataset. To do so, it creates an `EOPatch` per location with one feature for EOS data, another for soil property values, and some other features.

Before the modifications, five different `EOTask` were required. One to create `EOPatches` with specific EOS data (`SentinelHubInputTask()`), a second to append a location feature to each `EOPatch` (`AddFeatureTask()`), another that creates a mask to filter area (`VectorToRasterTask()`), an additional to combine the extracted band data into another feature of the `EOPatch` (`MergeFeatureTask()`), and a final one to save the `EOPatch` in the file system (`SaveTask()`). The original code can be found in `B.30`. Even though it worked, it lacked optimization and generalization. Precisely since:

1. Location was being saved as a feature dependent on position (`VECTOR_TIMELESS`), which implies soil properties would vary inside the same area;

2. `VectorToRasterTask()` unnecessarily computing a grid to discard band data from points outside of the location's area that was included posterior to Sentinel-Hub's extraction;

3. `SaveTask()` created `EOPatch` objects with no band data when the data collection had none to offer;

4. Only had access to S2 collection and forced a resolution of $10m$;

5. Unnecessary creation of a GeoDataFrame per row of a dataset, and its inclusion into the `EOPatch`;

6. `EOPatch` forced to have all column values present in the dataset.

Therefore, it was crucial to transform this method. Displayed in `B.31`, the reader can perceive that the feature task for location and sequential `VectorToRasterTask` were substituted to saving the `bbox` in the EOPatch (line 98); A new `EOTask` was created from the previously used `SaveTask()` that did not save any `EOPatch` with missing band data (line 80). Its creation can be viewed in `B.32`; No more GeoDataFrames are created in the main loop, saving the area correspondent to the EOS patch saved in a `bbox` variable (line 98); Only the soil properties specified in attribute `keep` are saved into the `EOPatch` (line 99). The user can specify which resolution, bands, maximum cloud coverage, and the time interval to extract EOS. `EOPatch` objects have access to a mapping that associates the band name and its channel index (line 76 and line 101). Finally, each `EOPatch` can be saved according to its identification value in the `id` column specified by the user and is not obliged to be an index (line 102). A coding example for the new downloader class can be found here[7].

5.2.2 Class: TSPatch

The `TSPatch` class is built to ease `EOPatch` representation and manipulation by making all information accessible and implementing useful functions. Its methods and attributes are displayed in Figure A.9.

5.2.2.A Modification: Attributes

Given the modifications conducted on the `EOPatch` creation process mentioned in Section 5.2.1, some attributes required modifications. For example, the location feature could be immediately accessed with `EOPatch.location`, now had to be extracted from `EOPatch.bbox`. So, the `patch` attribute was only renamed to `EOPatch` to clarify that it still holds an `EOPatch`. `id` was added to save the identification string for its `EOPatch`, `location` to permit a geographical recognition, `bands` to store the mapping of the band name and channel index present in the `EOPatch`, and `indices` to list the names of all indices calculated and present in this `EOPatch`. Most of these attributes were assigned at the class' constructor, which can be found in `B.33`

[7] Downloader, http://tinyurl.com/5kndzb94

5.2.2.B Modification: Simple Methods Alterations

Looking into its methods, most were an immediate consequence of adding the new attributes or from the modifications `EOPatch` suffered. `get_indices()`, `get_bands()`, `add_indice()`, `remove_indice()`, `get_location()`, `get_groundtruth_features()`, `get_groundtruth()`, `get_geometry()`, `get_date()`, and `get_groundtruth_value()` were all quick to implement and simplistic in function, as besides checking their arguments, they solemnly executed the low-level task of returning a saved value. Take for example the code written for `add_indice()` and `get_location()` found in `B.34` and `B.35`, respectively. The remaining methods also required alterations due to the modifications made in `EOPatch` creation, or for lack of generalization, or for missing some relevant aspects.

5.2.2.C Modification: `get_masked_region_values()`

Its original purpose was to return the area as a matrix of 0's with 1's per valid band pixel. However, since the feature that made that filter was removed from the `EOPatch` (reference to `VectorToRasterTask()` mentioned in Section 5.2.1), all `EOPatch`'s band data is valid. Thus, a modification was written, and if the present `EOPatch` does not contain a filter, it returns a matrix full of 1's. One can view the old implementation in `B.36`, and in `B.37`, the new code. Contrarily, method `get_masked_region_values()` did not suffer significant changes besides code restructuring, argument verification, and adding the possibility to get indices data. Ultimately, it maintained the same purpose and workflow as its original version, which extracted, in one or two dimensions, the masked pixel data from all bands. The old code can be viewed in `B.38` and the new version in `B.39`.

5.2.2.D Modification: Representing image in RGB

Method `represent_image()` was built to represent the EOS data from the `EOPatch` in three separated channels, allowing users to visualize its information in RGB. Like the prior, this function kept its purpose and only suffered a light code restructure, as the reader may find from comparing the old implementation written in `B.40` with the updated version presented in `B.41`.

5.2.2.E Modification: Selecting most valid EOS

Lastly, `_get_index_nearest_to_collection_date()` is a method that had to be extended. Bearing in mind that `EOPatches` band data have four dimensions, one temporal, another for channels, and the remaining two to create a matrix, and since one can select a temporal interval per location when extracting the EOS data, it is possible, and frequent with time intervals of 10 days, to extract more than one EOS per `EOPatch` from the `Downloader.download()` function. So, to avoid multiple EOS data connected to

one row of soil data, tstk always uses the one taken closest to the ground-truth's extraction date. However, this highly valuable process did not validate if the band data was unusable, which, unfortunately, happens frequently (some EOPatches captured from LUCAS were full of 0's). Thus, a modification was applied so that the method only returns the valid closest to date of extraction EOS data. Additionally, if all EOS data are invalid, an error is raised. The prior code can be read in B.42 and the updated code in B.43 A code example can be found in this file[8].

5.2.3 Class: Dataset

Dataset is a class that was written in tstk to represent an agglomeration of several TSPatches, plus to ease the process of executing important functions to the whole EOS data instead of individual computations. Its methods and attributes follow the right table displayed in Fig. A.10.

5.2.3.A Modification: Attributes

Before our implementation, this class totaled in four different attributes, them being eopatches_folder, _eopatches, df_eopatches and index_dic. The first one was to save the system's path of the directory with all EOPatches to be represented, whereas the second attribute keeps a collection of all TSPatches created from each EOPatch present in that folder. These variables were highly useful but seemed to lack an intuitive name, thus being renamed to eops_path and TSPatches when tstk suffered an update. df_eopatches, however, was an attribute that created a pandas.DataFrame to denote all EOPatches locations, which seemed partially complete. Therefore, to increase its usefulness, it was reprogrammed to include all information, with the exception of EOS data (identification, geometry, date, and ground-truth), present in each TSPatch found in TSPatches. This attribute was also renamed to gth_df. Lastly, the previous index_dic was a dictionary that mapped an index name to its calculation formula written as a string. This was replaced with mathematical expressions, using lambda functions. So, it was modified to solemnly include all indices appended to the existing TSPatch's data and consequently renamed to suit its new meaning (indices). To conclude, the remaining variables were extended for optimization and requirement purposes. lock is an attribute required to parallelize the time-consuming process of sequential TSPatch creation, bands was used to gather the band mapping from the TSPatches, and groundtruth_f is used to quickly access all soil properties present in these TSPatches. One can visualize the new constructor and parallelization method in B.44.

[8] TSPatch example, http://tinyurl.com/4jrru7ch

5.2.3.B Modification: Methods

Several methods were extended into tstk.Dataset's class, including five simple getter functions for the new or existing attributes, `add_indices()` (an update from `add_index()`), `remove_indices()` (due to transforming `index_dic` into a list), and `show()` to increase utility. Before our implementation, `add_index()` incremented the `index_dic` dictionary with a new formula but was limited to simple calculations with band names as strings. Additionally, calling `add_index()` alone produced no visible outcome since only `save_indices_to_eopatches()` applied these functions to each `TSPatch`, overwriting them. The function also lacked verification, risking invalid data in some `TSPatches` (see Section 5.2.2). We significantly transformed `add_index()`. Now, it receives a dictionary mapping the indice's name to its lambda formula, allowing all types of Python-supported mathematical expressions. Users must specify whether to keep calculated indices in the `TSPatch` object or save them with `EOPatches`. The function notifies users of mathematical errors, avoids them by adding a small random number, and ignores completely invalid `TSPatches` (full of 0's). The old code is in `B.46` and the new version in `B.47`.

To conclude this chapter, we address the last method included in this class, which is to extend the toolkit's utility and not to resolve any prior issues. It is called `show()` and just like the function `show_geometries()` added to `Downloader` class (referenced in Section 5.2.1), it is written to display the geographical shape and location of all `TSPatch`'s geometries that this `Dataset` class represents. A full code use case can be read here[9].

5.2.4 Class: Parser

The `Parser` is one of the extremely useful classes that come with tstk, enabling users to view and access partial or total information in `Dataset` in several different ways. From selecting the `TSPatches`, to the ground-truth features, `Parser` could output a `pandas.DataFrame` follows the user requirements without needing extra computation. Its design follows the table displayed in Figure A.11.

5.2.4.A Modification: Attributes

Taking a closer look at its prior attributes, `subset` was created to save the collection of intended `TSPatches`, `indices` and `bands` are both lists with names of EOS features of interest to be extracted from the `Dataset`, `_dataset` to keep the `Dataset` object used as the source of data, `save_dataframe` is a `pandas.DataFrame` created with the intended characteristics, `dataset_extra_data` a list of ground-truth columns chosen to be viewed, and `image_identifier` the column used to identify the `TSPatches`. All these attributes suffered minor changes, as they save crucial information for the `Parser` to function correctly. Therefore, unlike the previous sections, there were no changes that justify their mentioning, besides a simple adap-

[9] Dataset, http://tinyurl.com/2s3r245k

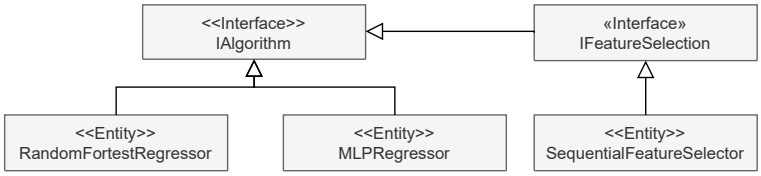


Figure 5.3: `Algorithms` module before restructuring. Note: This is a shortened representation. For more details view Figure A.12.

tation of `bands` to include the mapping created in Section 5.2.1, and the renaming of others to decrease doubt on their purpose.

5.2.4.B Modification: Methods

Regarding its methods, the main difference comes with the addition of several getters and setters to increase the class' malleability, being `create_df()` and `convert()` the result of a simple code cleaning of `create_dataframe()` and `convert()`, plus the addition of more function arguments to allow a wider range of possibilities. Furthermore, being such a useful method, `show()` was also extended to this class, allowing the user to geographically visualize all `TSPatches` in `subset`. Here[10] the reader may find a code example of this class.

5.3 Module: Algorithms

The `Algorithms` module was designed to represent any computation process applicable to ML problems that require a training and testing step. It suffered several, if not the most, changes in its core that not only affected its structure and the algorithms it provided but also brought SSL models. These modifications can be better perceived by comparing the prior illustration found in Figure 5.3 with what's presented in Figure 5.4.

5.3.1 Modification: Structure

The main structural difference is the transformation of `IAlgorithm` from an interface that represents all algorithms that require a training and testing step to an abstract class that generalizes any predictive ML model. Such modification led to the addition of several attributes common in any ML model, independently of its nature. This abstraction permits an easier implementation of new algorithms and a simpler representation of the already developed ones since most of the code is written on the upper class. In fact, as one can foresee from Figure A.13, `Regressors` and `Classifiers` have only one

[10] Parser, http://tinyurl.com/485cssy5

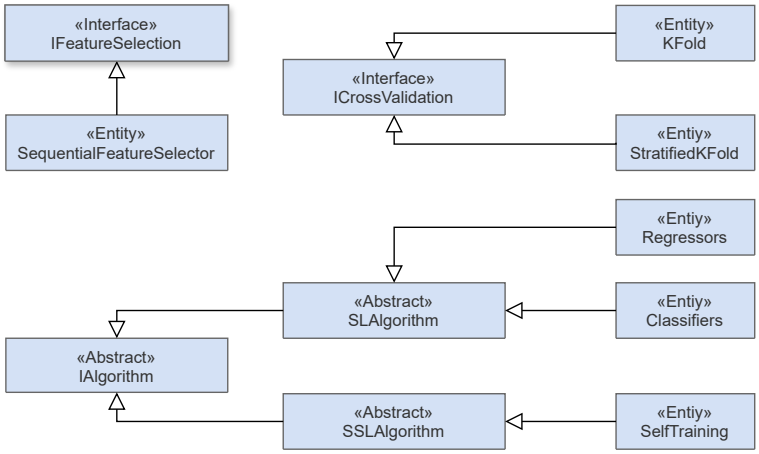


Figure 5.4: `Algorithms` module after restructuring. (`Classifiers` and `Regressors` represent several implemented models of each type, and are not a real class in the toolkit). Note: This is a shortened representation. For the more detailed one focus on Figure A.13.

attribute and three methods, but all make use of the sixteen attributes present in `IAlgorithm` class. Consequently, it also led to the disconnection of `IFeatureSelection` from class `IAlgorithm`. Moreover, as SSL models were always a mandatory addition, two other abstract classes were added: `SLAlgorithm` and `SSLAlgorithm`. This incorporation serves the same purpose as the abstraction of `IAlgorithm`, by generalizing all SL and SSL models present in the toolkit. The final structural modification was relocating class `ICrossValidation` from the `Performance` module to this module. It was done to maintain structural coherence since `ICrossValidation` are also algorithms, and not tools to measure model's performances.

5.3.2 Modification: Algorithm Availability

Before our modifications, tstk contained five classic ML regressor models, plus an additional feature selection algorithm named `SequentialFeatureSelector`. Thus, it lacked crucial requirements for our investigation: Classifiers and SSL. As mentioned in Section 3.3.2, `scikitlearn` is a reliable source of ML algorithms, USB's interface is incompatible with tstk and LAMDA is enormously lacking validation. Therefore, we naturally concluded the selection of `scikitlearn` as the single source of SSL code. Therefore, a total of six SL classifiers, one SSL classification class (named `SelfTraining` that can be used with any classifier from `scikitlearn`), and another cross-validation algorithm represented in `StratifiedKFold` class were added to this toolkit.

5.4 Module: Performance

The `Performance` module was designed to evaluate the performance of any ML model. It included an interface (`IMetrics`) to generalize any class measuring model execution, and another interface (`ICrossValidation`) for creating cross-validation algorithms. However, the module lacked a class to assess classifiers and misplaced the cross-validation model. These issues necessitated further modifications. The following subsections address these problems, and Figure A.14 visualizes the changes made to resolve them.

5.4.1 Class: Result

The `Result` class is not a new implementation to tstk, as it was already found in the toolkit. Considered to be unrelated to any other module, this class was placed inside of a utility directory, which included several other classes/files that were useful but lacked correlation to the other modules. Its former state can be foreseen in Figure 5.2. Through our perspective, however, this specific class that agglomerated all the information regarding a model's result and evaluation made complete sense to be in the `Performance` module. After all, it is the object from which all evaluations will be conducted, and that contains the model's result.

Besides its relocation, it only suffered some light modifications, them being the addition of attributes crucial to measure the model's performance and some other required for classification measurements, the alteration of the private method `__repr__()` to output a better representation of the object's state, and the inclusion of an auxiliary method `conf()` which calculates the confusion matrix for the `Result` object (only functional for `Result` objects created from classifiers).

5.4.2 Class: IMetrics

Regarding our second transformation, the `IMetrics` interface was converted to an abstract class as both the regression and classification classes use the methods it implements. Thus, we avoid code duplication and invalidate the creation of a measurement class without the specification of the model's nature. So, it suffered a conversion but kept its original purpose. Aligned to this and already mentioned, a new class was incorporated that specifies the evaluation of classifiers (`ClassifierMetrics`). This new class copied the structure and logic behind the original `RegressionMetrics` class, which did not suffer any alterations. Therefore, it simply consists of a listing of methods, each making use of a `sklearn` function purposely developed to calculate a single measurement. Take, for example, `cmd_acc()`, `cmd_prec()` and `cmd_ae()`, which compute a result's accuracy, precision, and absolute error, respectively. These `IMetrics` sub-classes are not bound to a single `Result` object, and so one `IMetric` object can measure several `Result`.

5.5 Class: Experiment

To finalize this section, we take a closer look into the `Experiment` class. With it, users are able to conduct experiences by evaluating a given number of tstk models on their predictions made for a singular set of data. The data is also defined by the user through a `Parser` object. The true potential of this class comes with the simpleness and variety of configurations users have to customize their ML experiences, which are made by changing the `Experiment` object's parameters. Thus, through this class, tstk users have a quick and straightforward option to test their ML models.

However, some changes still had to be made, as this class presented several unnecessary attributes and methods which were outshining its potential and over-complicating its usage. Moreover, its computation followed a logical path that could also be improved. From Figure A.15, one can visualize the alterations performed. Therefore, and just like the previous subsections, the following subsections enlighten the reader with how these problems were surpassed.

5.5.1 Modification: Structure

As mentioned before, the first alteration aimed to reduce the class' complexity by reducing the number of attributes and methods. From our perspective, most attributes were labeled as unnecessary, and so considered to be overloading the class. Take, for example, attribute `cross_validation_complete`, which was a boolean variable that became true when the experiment finished running the cross-validation algorithm. It could be removed since these algorithms must only be run once and always before the models' execution. So, its existence is unjustified with properly implementing the class's logical flow. Most attributes suffered changes similar to the one just mentioned, which led to the enormous reduction from 21 attributes to seven. On the contrary, all seven new attributes were considered necessary as they represent information that can not be omitted or obtained through other variables. `name` represents the name defined by the user to identify the `Experiment`, `parser` and `fs` serve to keep the objects used during the experiment, `models` a list of all `tstk.IAlgorithm` selected to be evaluated, `verbose` an additional boolean that, when true, outputs execution messages, `models_type` to indicate which models are classifiers, and `results` to save a performance level per model executed. The same goes for its methods. Most were removed, and others were modified, resulting in one private and two public methods. `_check_args()` is an internal method that verifies if all chosen user configurations are valid, `execute()` to perform the experiment and `calculate_metrics()` to evaluate each model result.

5.5.2 Modification: Logical flow

To better grasp the restructuring process that took part in the enhancement of this class' logical flow, the reader may compare the execution process portrayed in Figure A.16 with Figure A.17. On it, one can

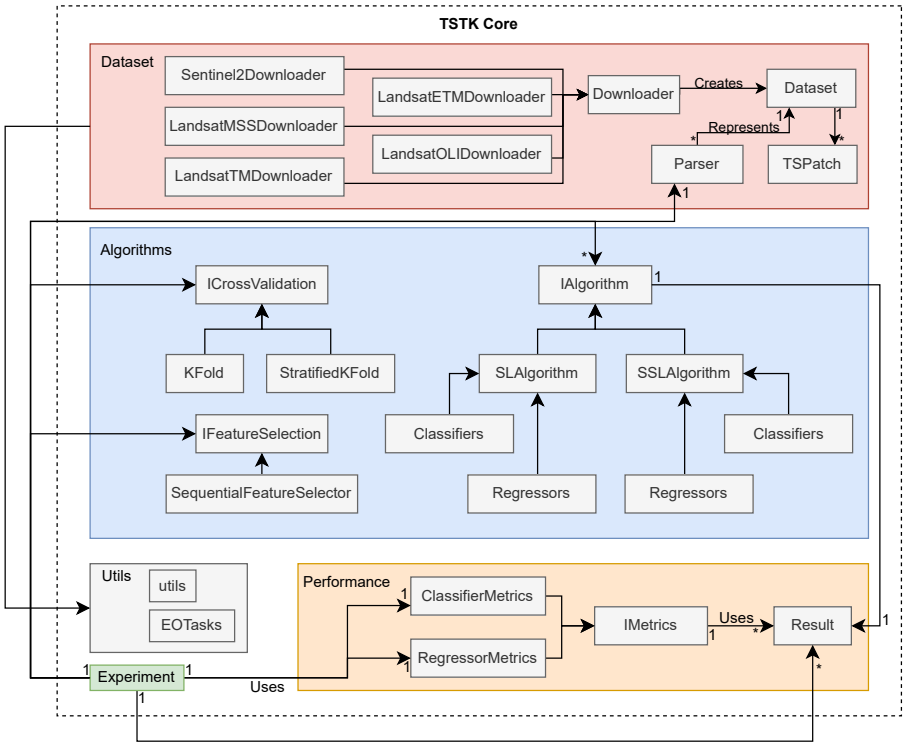


Figure 5.5: tstk core after restructuring.

perceive that the prior state required a change, as it lacked an optimal solution. With our new approach, all models manipulate their data according to their specific needs when the class is constructed. When the user runs `execute()`, it now has the option to save the `Result` objects in a `pickle` file, and all computation is done for each model sequentially. Some other improvements were made that are not perceivable from the presented figures, such as each model only saving its best result.

5.6 Summary

In conclusion, tstk suffered many modifications which increased its scope, functionality, and utility, reduced its complexity, and above all, made it more suitable to conduct ML investigations on EOS data. Attributes were changed, logical flows were optimized, several new features were included. Combined, these alterations changed the toolkit's main structure, which is represented on this final diagram Figure 5.5

6

Semi-Supervised Learning: Case Study Specification

Contents

In this chapter, the reader may find the pre-processing methods conducted on our final dataset to assess its quality and prepare its data for the upcoming experiments, followed by a second section that presents all the different experiments that will be executed to understand and evaluate how SSL models compare with SL on nutrient prediction.

6.1 Data Pre-Processing

As mentioned in Chapter 4, our final dataset corresponds to two versions of the LUCAS dataset merged with the LUCAS topsoil, after the exclusion of all locations with 'Artificial Land', 'Wetland' and 'Water' land cover, only taking into account P nutrient values and an additional binary classification conversion according to general P fertility concentration limits ($10.9ppm - 21.4ppm$). The first version, which will be referenced as `dataset`, agglomerates 16.131 data points. 11.813 points are X_l and the remaining 4.318 X_u, being P majorly labeled as infertile ($66,20\%$). On the other hand, the second version reduces the dataset to all data points classified as 'Common wheat'. It covers 1.218 labelled plus an additional 132 unlabeled data points and possesses an even higher unbalance state, being 74% of its P values infertile (Table 4.6). It will further be referenced as `dataset_wheat`.

6.1.1 Data Distribution

Our first pre-processing approach validates the quality of EOS data obtained from the Sentinel's Hub API, which, as explained previously, is how tstk has access to satellite data. Such a decision was made after coming across some `EOPatches` that presented impossible (or that should be impossible) band values. Therefore, to investigate the matter, a scatter plot will be drawn per remote-sensing feature, and all values outside of the feature's range will be counted. Being SVI calculated from band values, it seemed natural to follow the same approach. From the broad selection of SVI chosen for our experiments, only $NDVI$, $NDMI$, $NDRE$, $NDSI$, $NDVI(780,670)$, $GNDVI$, $NDSI(R523,R583)$, GRI, CL and $DSI(R1705,R1385)$ present functions with limited scopes, and so, only these will be investigated. These plots will present a red line that symbolizes the feature's function range.

To finalize our data distribution assessment, we will closely examine the number of outliers per feature present in `dataset`. Such measurement comes from the ongoing discussion that they can negatively affect ML predictions. To do so, simple box plots will be drawn per remote-sensing feature and also count the number of data points that satisfy the outlier condition 2.8.

6.1.2 Feature Selection

Bearing in mind its purpose, feature selection gains value to this research as one can use it to gain insight into the potential that our EOS data has to predict P density. Thus, after assessing the dataset's distribution, we intend to discover a generalized set of features that have the highest impact on P prediction. To do so, we will independently execute a series of feature selection algorithms on `dataset` to obtain a range of strongest features according to each model. In this feature selection pipeline, the two of the three previously mentioned models in Section 2.3.5 (PCA and RFE) will each designate a set of features as the most relevant for P prediction, and after their completion, we intercept the three different selections and obtain a final set of most important features, valid to all models.

Our approach is straightforwardly applicable to PCA and RFE, as both algorithms merely output a set of features they believe to have the highest importance. $Pearson$ correlation, however, does not return a set of features but the correlation's intensity per feature. Such capability can and will be explored differently. It will be used to calculate the connection that all features have with each other, and if they share high values (more than 90%), we keep only one.

So, with this combination of methods, we are able to extract the features that both models agree to be the most important, removing redundant data, and hopefully increasing our ML model's accuracy and reducing our experiment computation complexity.

6.2 Soil Nutrient Classification with ML

In this section, we dive into the various experiments created to evaluate our ML models. Firstly, we introduce a set of specifications that all experiments share, from mentioning the parameters chosen for each model, explaining the hyper-parameter tuning process to the procedures used during training and testing, and revealing what performance metrics were used to evaluate their results. Furthermore, we present and justify each of the three main experiments that were selected for our research.

6.2.1 Algorithms, Parameters & Hyper-parameters

As revealed in Section 2.3.2, our investigation focuses on five different SL models (KNN, CNB, RF, GradBoost, AdaBoost) and their SSL self-training versions. Each of these models has a considerable amount of variations due to their broad variety of parameter values, which are not to be discarded but instead to be examined as all have the potential to score the best results. So, a set of each model's parameters will undergo a parameter optimization process, and Table 6.1 describes which. To do so, each model will be executed at 100 times, where their hyper-parameter values change at each iteration. This leads to a total of 100 different performance results per model, but we will consider the model's

version with which it achieved the highest accuracy.

Model	Hyper-parameter	Description
KNN	$n\_neighbors$	The number of neighbours to be considered.
	$weights$	Weight function used in prediction.
	$algorithm$	Algorithm used to compute the nearest neighbors.
	$leaf\_size$	Parameter for algorithms BallTree and KDTree.
	$metric$	Metric used to measure distance.
	p	Parameter for the Minkowski metric.
CNB	$alpha$	Additive smoothing parameter.
	$force\_alpha$	Limits alpha value to $1e^{-10}$.
	$norm$	Performs a second weight normalization.
RF	$n\_estimators$	The number of trees in the forest.
	$criterion$	The function to measure the quality of a split.
	$max\_features$	Number of features to select the best split.
GradBoost	$learning\_rate$	Multiplicative factor for the leaves values.
	max_depth	The maximum depth of each tree.
	$min\_samples\_leaf$	The minimum number of samples per leaf.
AdaBoost	$estimator$	The base estimator.
	$n\_estimators$	The maximum number of estimators.
	$learning\_rate$	Weight applied at each boosting iteration.
	$algorithm$	Specifies the boosting algorithm.

Table 6.1: Model hyper-parameters and their descriptions. These are subject to parameter optimization.

6.2.2 Training and Testing

Concerning the training and testing procedure exposed in Section 2.3.1, this subsection sheds light on some specifications that both these processes were forced to have in our experiments.

Addressing first the training step, two worth mentioning situations had to be taken care of: the absence of ground-truth area for some data points, plus the incapability of our ML models to work with two-dimensional features. As one of the datasets (LUCAS Topsoil 2018) used to create our final `dataset` did not have a ground-truth area for each soil data, but simply a geographical point from which we are unable to extract EOS data, we were forced to transform them into a larger area that surrounds its coordinates. To do so, we made use of the tstk newly-built method `calculate_area()` with which we calculated a squared ground-truth area of $60m^2$ centered in these data point locations. Regarding our model's inability, we had to convert all our two-dimensional EOS data to a single value, which was accomplished through an averaging of all pixels. So, our models will no longer receive a matrix of values for each feature but a single value representing all pixels. Thus, X reduces from three dimensions to two dimensions.

Moving to the testing step, we performed cross-validation on all experiments. Discarding such an approach would result in bad performances, not only due to our dataset's size but also to their class distribution. Firstly, it was considered to use the `KFold` algorithm, but it completely randomized what

each fold would receive, leading to folds with different balance states. Therefore, we opted to conduct our experiments with a variation of this algorithm, named `StratifiedKFold`. Also built in the `sklearn.model_selection` python toolkit, this cross-validation version ensures that each fold preserves the same percentage of class distribution.

6.2.3 Performance Metrics

In order to evaluate our experiments, we initially intended to make use of the classification metrics previously mentioned in Section 2.3.4, them being accuracy, precision, recall and f1-score, which, as written in the chapter, serve different purposes. Translating to our investigation, accuracy represents the percentage of data points correctly classified, precision the amount of correct 'fertile' predictions on the total of its predictions, and recall the share of data points correctly labeled as 'fertile' in all 'fertile' data points. Bearing in mind that our goal is to have a general evaluation, that both recall and precision share a high value of information, and that f1-score combines both, we will make use of this measurement (f1-score) as a tool to assess our models. Furthermore, bearing in mind that accuracy is extremely fragile to imbalanced datasets, it will be changed to one of its versions, which is purposely built to overcome such a state. It is named balanced accuracy and is calculated as shown in Eq. (6.1). Translated to our research problem, P and N correspond to the total number of 'fertile' and 'infertile' data points in Y_{test}.

$$balanced\_accuracy = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right) \tag{6.1}$$

6.2.4 Experiments Description

Our goal is to measure if there are some advantages of using SSL models compared to the default SL versions. This is achieved by comparing their performances when trained and tested on the same dataset. In this paper, this translates to a direct comparison between the overall results achieved with the previously mentioned SL models and their Self-training versions after being executed on both `dataset` and `dataset_wheat`.

From selecting which bands and SVI to feed into the models, to overcoming or ignoring the classes' unbalanced state, all would undoubtedly portray different results when applied to or omitted from our experiments. Therefore, not knowing which adaptation outputs the best result, we decided to evaluate these several scenarios by creating a variety of experiments. The experiments built are listed below:

1. An experiment that uses all twelve bands plus the additional twenty-eight SVI.

2. Another experiment that only utilizes features that have been proven, according to literature, to have adequate results when predicting P values.

3. And a final one using all features which were outputted from a series of feature selection algorithms.

The first one was built as a baseline to investigate how the models behaved when fed with all our EOS information. On the other hand, the remaining were developed to assess if general accuracy increased when only considering features proven to be correlated to P. The second one made use of features proven by research papers (domain-driven approach), whereas the third one used the $Pearson\ correlation$, PCA and RFE algorithms (data-driven approach).

Inspired by five articles, the second experiment selected fifteen SVI and eleven bands to feed into our models. [50], [51] and [52] defended the following SVI: $NDVI$, EVI, $NDMI$, PVI, $SAVI$, $TSAVI$, CL, $Chlgreen$, CIG, $FE2$, $GRNDVI$, $NDVI$, $GNDVI$, $NDSI(R523, R583)$ and GRI; and articles [53] and [54] justify using all available bands with exception to $SWIR12$.

The features elected for the third experiment require evaluating our results, so this chapter will only identify which were selected. Its explanation will be shown further in Chapter 7. Thus, both experiments discard all features they consider to be unrelated to our target but diverge on how they establish this association.

Nevertheless, even though cross-validation is employed, our datasets were still heavily unbalanced, which could prejudice our results. Therefore, it was decided to include experiments specifically developed to mitigate this hurdle. More precisely, each of the experiments mentioned above will also be executed after equalizing the data records labeled as 'infertile' to the ones classified as 'fertile'. This process is called undersampling, and after its application on `dataset`, the number of 'infertile' records reduces from 7.772 to 3.968, thus changing to have a total of 7.936 all labeled data points with an equal classification distribution.

This approach removes all previous X_u, and so, not only does a comparison between SSL and SL, but also fails to study the impact the amount of X_u has on SSL models. Thus, another set of experiments was added to our execution plan where `dataset` is undersampled, and a percentage of its new X is unlabeled. So, six new experiments were added to our investigation. All use the `dataset` after being undersampled, but vary on the percentage of X_u. Only 5%, 25%, 50%, 75% and 100% of their X are kept as labeled data.

All our experiments are represented on the combination of Table 6.2 and Table 6.3. The first one contains the core information related to all experiment variations. It displays the dataset used as a source of data, if undersampling was performed, the number of labeled and unlabeled data, and the classes representation. Each row is not associated to a single experiment, but to the conditions applied to each of the three experiments enumerated as 1, 2 and 3. The second table is shown to easily identify which features the three main type of experiments will feed into the models.

Experiment variations	*Dataset*	*Under-sampling*	X_l	X_u	*Fertile %*
`All-Features`	`dataset`	X	11813	4318	33.78%
`Wheat`	`dataset_wheat`	X	1218	132	25.53%
`Undersampled-5%Labeled`	`dataset`	✓	400	7582	50.0%
`Undersampled-25%Labeled`	`dataset`	✓	1996	5986	50.0%
`Undersampled-50%Labeled`	`dataset`	✓	3992	3990	50.0%
`Undersampled-75%Labeled`	`dataset`	✓	5986	1996	50.0%
`Undersampled-100%Labeled`	`dataset`	✓	7982	0	50.0%

Table 6.2: Information regarding the eight variations of experiments that will be executed for each of the three core experiments mentioned in 1, 2, and 3.

Features	*Exp.*1: *All-Features*	*Exp.*2: *Literature-Features*	*Exp.*3: *Feature-Selection*
Ultra-blue	✓	✓	✓
BLUE	✓	✓	
GREEN	✓	✓	
RED	✓	✓	
VNIR5	✓	✓	
VNIR6	✓	✓	
VNIR7	✓	✓	
VNIR8	✓	✓	✓
VNIR8a	✓	✓	
SWIR9	✓	✓	✓
SWIR11	✓	✓	
SWIR12	✓		
NDVI	✓	✓	
EVI	✓	✓	
NDMI	✓	✓	
PVI	✓	✓	
SAVI	✓	✓	
TSAVI	✓	✓	✓
CL	✓	✓	
Chlgreen	✓	✓	
CIG	✓	✓	
FE2	✓	✓	
TCARI	✓		
MCARI	✓		✓
CCCI	✓		
NDRE	✓		
CIrededge	✓		
CVI	✓		
GSAVI	✓		
MSAVI	✓		
RDVI	✓		
WDRVI	✓		✓
NDSI	✓		
RSI(1385, 1705)	✓		
DSI(R1705, R1385)	✓		
NDVI(780, 670)	✓		
GRNDVI	✓	✓	
GNDVI	✓	✓	✓
NDSI(R523, R583)	✓	✓	
GRI	✓	✓	

Table 6.3: Features used for each of the three main experiments.

7

Semi-Supervised Learning: Results and Discussion

Contents

This second to last chapter begins with a straightforward presentation of all the results obtained from the pre-processing procedures detailed in Section 6.1 along with a discussion of what they unveiled. The second subsection follows the same structure but regarding the experiments prepared and mentioned in Section 6.2. Finally, the chapter ends with a conclusion to our research.

7.1 Data Pre-Processing

7.1.1 Data Distribution

Recalling Section 6.1.1, our fist pre-processing evaluation focused on the dataset's quality. More specifically visualizing and counting the number of data points per feature that falls from its function range, and calculating the total amount of outliers they possess. Such investigation made use of a scatter-plot with a red dashed line symbolizing the feature's scope, plus a box-plot graph drawn per feature.

Bands	Number of invalid
Ultra-blue	74
BLUE	59
GREEN	48
RED	43
VNIR5	62
VNIR6	53
VNIR7	46
VNIR8	44
VNIR8a	38
SWIR9	316
SWIR11	1
SWIR12	0

SVI	Number of invalid
NDVI	0
NDMI	0
NDRE	13
NDSI	4
NDVI(780, 670)	4
GNDVI	0
NDSI(R523, R583)	0
GRI	0
CL	0
DSI(R1705, R1385)	0

Table 7.1: Both tables present the number of EOPatch from dataset with values outside of their feature's scope. Band SWIR9 and SVI NDRE present the most number of invalid data. Note: Invalid values are higher than 1.0 or lower than 0.0 for bands. However, SVI vary on their scope depending on their formula. CL and DSI(R1705, R1385) invalid values are higher than 2.0 or lower than -2.0. The remaining SVI are only correct when between -1.0 and 1.0. Only the SVI with limited range were investigated.

From observing the series of plots displayed in Fig. A.18 and from the information presented in Table 7.1, we can firmly state that almost no bands possess invalid data. In fact, from the 16.033 data points, SWIR9 had the worst representation, but its 316 data points only portray $1,97\%$ of its data. Additionally, bearing in mind that all SVI are calculated from band values, one can theorize that it shares the same optimistic validation. Thus, we followed the same approach and discovered that in fact SVI also present coherent values. From the ten SVI investigated, only three presented erroneous data points. NDRE had the highest amount of invalid data but being a slight 13 EOPatches. Their scatter-plots can be observed through Fig. A.19.

Even though a small amount of invalid values might raise doubts, one must keep in mind that it is

not uncommon for remote sensing sensors to read incoherent data. S2 sensors are prone to reflections on bright areas such as snow, which are reported to have values higher than 1.0. So, having `dataset` nothing but a slight amount of possible inconsistencies, we can clearly state that it is a resourceful fount of high quality EOS. Such guarantee strengthens the experience's contribution and results, and insures reliability in our python toolkit.

Bands	Number of outliers
`Ultra-blue`	1345
`BLUE`	1205
`GREEN`	1131
`RED`	865
`VNIR5`	935
`VNIR6`	591
`VNIR7`	468
`VNIR8`	502
`VNIR8a`	418
`SWIR9`	799
`SWIR11`	349
`SWIR12`	359

SVI	Number of Outliers
`NDVI`	42
`EVI`	165
`NDMI`	47
`PVI`	496
`TSAVI`	308
`CL`	293
`Chlgreen`	441
`CIG`	150
`FE2`	646
`TCARI`	174
`MCARI`	585
`CCCI`	739
`Clrededge`	143
`CVI`	380
`GSAVI`	119
`RSI(1385, 1705)`	126
`DSI(R1705, R1385)`	420
`NDSI(R523, R583)`	662
`GRI`	149

Table 7.2: Both tables present the number of outliers on `dataset` per feature, where band `Ultra-blue` and SVI `CCCI` present the highest amount of outliers. Note: All SVI containing less than 100 outliers ($0,62\%$) are not displayed.

Regarding the number of outliers in our dataset, Table 7.2 reveals that both band and SVI features contain some data points that fall into Equation (2.8). Maxing at 1345, band `Ultra-blue` is the feature with the highest percentage of outliers, and even though it is a considerable amount of `EOPatches` with relatively abnormal data points, they only represent $8,39\%$ of its distribution. The remaining bands have approximately 7%, 5%, 3% and 2% of their data as outliers. All bands box-plots are visible through Fig. A.20. Interestingly, SVI present lesser values, being $4,61\%$ the highest amount reached with `CCCI`, and the remaining having lower than 3%. These results are drawn on boxplots present in Fig. A.21 and Fig. A.22.

When discussing our outlier values, one should keep in mind its core concept. Outliers represent data points located far from the distribution of the other values. Thus, its definition is relative to the dataset, and does not take into consideration a real representation of data. Specifying to our dataset, it is know that remote sensing data diverges from snow regions to dense forests, and since `dataset` contains a slight representation of the first, bands and SVI taken from snowy regions will be labeled as outliers, even

though they are correct. The same goes for any other unique land cover that is negligible represented. I.e. `dataset` may present several outliers that would not appear if it were equally distributed. Bearing in mind that our dataset is spread through several regions of Europe, where each varies in land cover and sun ray inclination, these outliers results do not suffice to claim that `dataset` is an unreliable source. In fact, when we restrict our dataset to `EOPatches` taken from the Iberian peninsula, we get 4039 data points from which band `Ultra-blue` maintains the highest number of outliers (234), but its percentage drops to $5,79\%$. Furthermore, counting the number of outliers present in `dataset_wheat` taken only from the Iberian peninsula, from the 271 data points left only 16 `Ultra-blue` values are labeled as outliers ($5,9\%$).

So, we can conclude that our `dataset` does not have an alarming amount of outliers, since some of those are due to its unbalanced representation of land cover. From our results and thoughts, `dataset` strikes as a reliable, high quality and diverse source of EOS data to train ML models.

7.1.2 Feature Selection

Moving to the last procedure performed to analyse our data, we made use of a combination of feature selection models to elect which are the most accurate at predicting P, and then utilize the $Pearson$ method to remove the elected features that share the same information. Practically speaking, this process of feature selection through the combination of three distinct algorithms goes as follows: Firstly we intercept the features selected from PCA with those outputted from RFE, and from this resulting set, remove any combination of features that share a $Pearson$ correlation above 0.9. By the end of this procedure, we extract a set of optimal features to predict P, which make experiment 3.

The first model to be executed was PCA, which considered bands `Ultra-blue`, `SWIR9` and `VNIR8`, plus SVI `GRNDVI`, `RSI(1385, 1705)`, `CVI`, `FE2`, `CCCI`, `CL`, `EVI`, `CIG`, `Chlgreen`, `NDSI(R523, R583)`, `NDSI`, `MCARI`, `GNDVI`, `TSAVI`, `WDRVI`, `NDVI(780, 670)` and `GSAVI` to be the twenty most important features to represent our data. RFE however, discarded bands `BLUE` and `SWIR12`, plus SVI `CVI`, `EVI`, `FE2`, `PVI`, `CIrededge`, `NDSI`, `GRI`, `CCCI`, `CL`, `RSI(1385, 1705)`, `Chlgreen`, `NDSI(R523, R583)` and `CIG`. So, by intercepting these two, we reach a set of features which both algorithms agree to be the most crucial variables to predict P. They are bands `Ultra-blue`, `VNIR8` and `SWIR9`, plus SVI `GNDVI`, `GRNDVI`, `GSAVI`, `MCARI`, `NDVI(780, 670)`, `TSAVI` and `WDRVI`.

Subsequently, we deviated to the $Pearson$ method to measure the correlation values that these new set of features share between themselves, in order to remove those that transmit the same information. This way, we reduce the amount of data our models have to train with, and so optimize the experiment. We analysed all bands and all SVI, from which we obtained both matrices displayed in Figure 7.1.

Taking a look into our results we observe that bands `Ulta-blue`, `Blue`, `Green`, `Red` and `VNIR5` share a correlation above 0.9. The same is seen for groups `VNIR7`, `VNIR8` and `VNIR8a`, plus `VNIR6` with `SWIR9`,

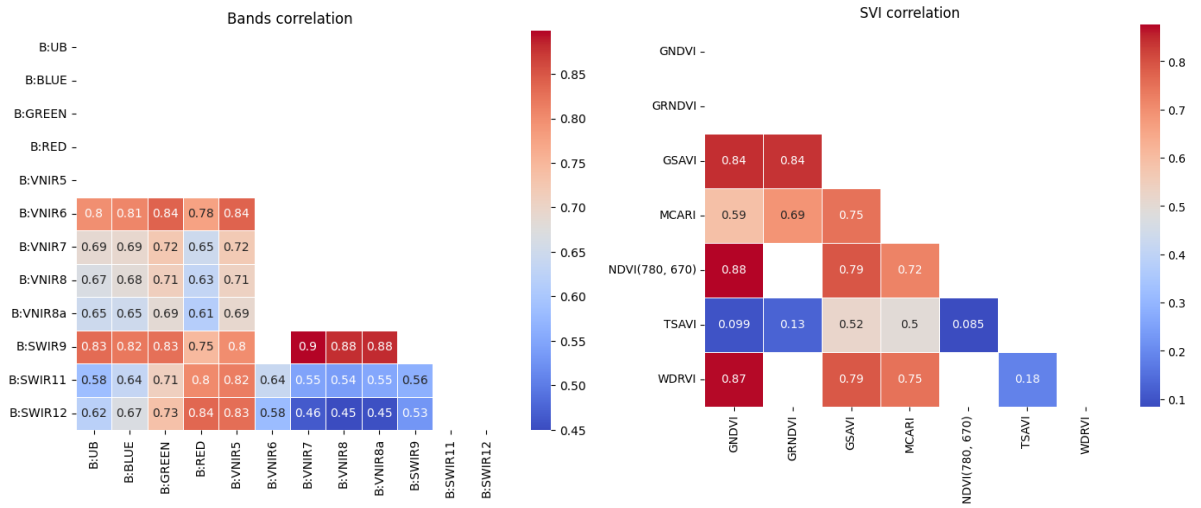


Figure 7.1: Both matrices portray the correlation between features present on `dataset`. The left one represents all bands, and the other, for simplification, only shows the values shared between the SVI elected by PCA and RFE. Note: Squares omitted present a correlation higher than 0.9. To view the correlation values for all indices, check Figure A.23

and finally `SWIR11` with `SWIR12`. Having identified which groups of features share the same information, we can apply these discoveries to the set of features mentioned before. Recalling the set of bands elected (`Ultra-blue`, `VNIR8` and `SWIR9`) none are more than 90% correlated, as `Ultra-blue` shares 0.67 with `VNIR8` and 0.83 with `SWIR9`, and `VNIR8` shares 0.88 with `SWIR9`. Therefore, we conclude that none of the bands resultant from the interception of PCA and RFE are correlated, and so all are kept on the final set of features for experiment 3.

The same methodology was applied to our SVI features, where, as perceived from the correlation matrix on the right of Figure 7.1, some present a strong correlation between themselves. From this result, we conclude that `GRNDVI` is strongly correlated to `GNDVI`, `WDRVI` and `NDVI(780, 670)`, and that the last two are also connected between themselves. Thus, to avoid extra computation of features that transmit the same information we discard `GRNDVI` and `NDVI(780, 670)`.

Therefore, we reach the final set of features that were previously presented in Table 6.3

7.2 ML Experiments

7.2.1 Performance per sub experiment

The first evaluation conducted on our experiments focused on discovering the general performance levels our algorithms had on P prediction. It was also in our interest to discover if any of the major Experiments (1, 2, 3) supplied better conditions to solve the task at hand. Therefore, these results were visualized by averaging the balanced accuracy and f1-scores for each of the seven sub experiments,

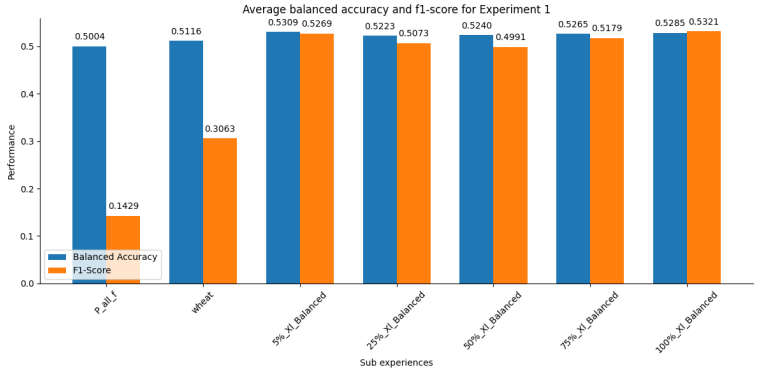


Figure 7.2: Balanced accuracy and f1-score average per sub experiment conducted on the experiment 1. The highest balanced accuracy was achieved with sub experiment $5\%_X_l_Balance$d, but it was $100\%\_X_l\_Balanced$ that reached the biggest f1-score. Note: Values were calculated by averaging all models.

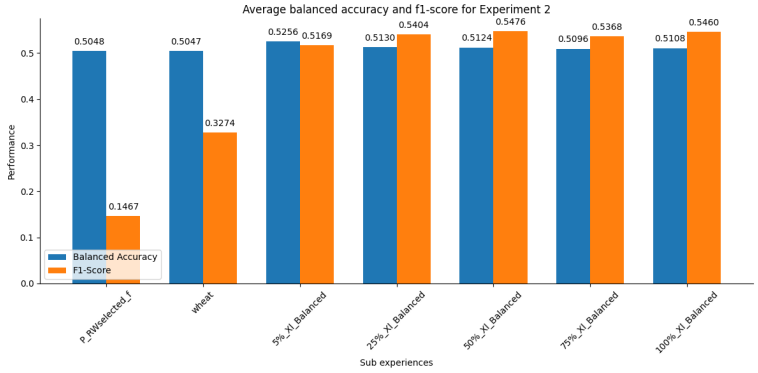


Figure 7.3: Balanced accuracy and f1-score average per sub experiment conducted on the experiment 2. The highest balanced accuracy was achieved with sub experiment $5\%\_X_l\_Balanced$, but it was $50\%\_X_l\_Balanced$ that reached the biggest f1-score. Note: Values were calculated by averaging all models.

independently of the algorithm. This measurement was separately conducted per major experiment, thus originating three distinct bar plots Fig. 7.2, Fig. 7.3 and Fig. 7.4. These graphs contain two bars per sub-experiment. One represents average balanced accuracy and another the average f1-score.

Unlike what was thought, the pre-selection of features gave little to no performance gain, as the highest average balanced accuracy throughout the three experiments (0.5318 achieved with Experiment 3, as displayed in Figure 7.4) was slightly above the highest value achieved with the experiment that considered all the twenty-eight features (0.5309 on Figure 7.2). The same conclusion can be derived from looking into the f1-score values. It was with Experiment 2 that we achieved the highest value of 0.5476, but Experiment 1 also came close to this number, peeking at 0.5321, which is less than 2% apart.

Furthermore, another observation can be made from these bar plots. Theoretically proven before, and clearly represented in our results, f1-scores are extremely dependent on the dataset's balance state. Such conclusion can be derived from the gain it had when comparing unbalanced datasets sub

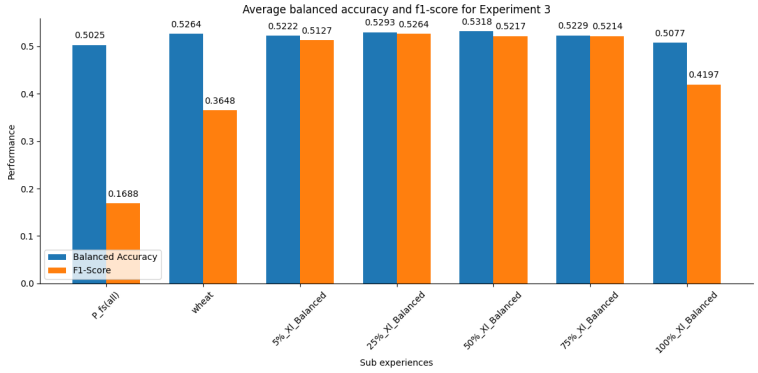


Figure 7.4: Balanced accuracy and f1-score average per sub experiment conducted on the experiment 3. The highest balanced accuracy was achieved with sub experiment $50\%\_X_l\_Balanced$, but it was $25\%\_X_l\_Balanced$ that reached the biggest f1-score. Note: Values were calculated by averaging all models.

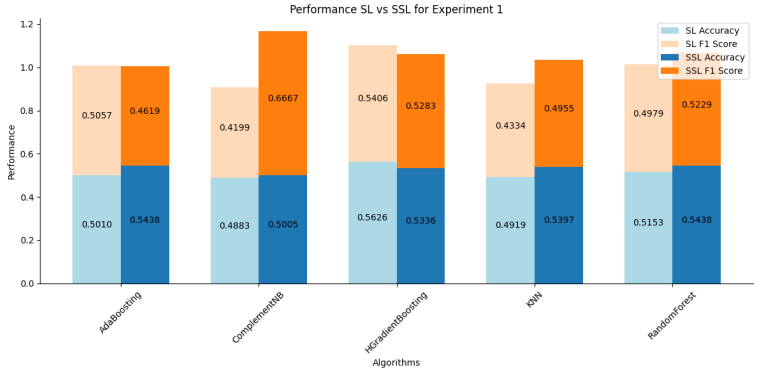


Figure 7.5: Balanced accuracy and f1-score comparison between each model and their SSL adaptation. These values only take into account the sub experiment `Undersampled-25%Labeled` of experiment 1. Note: Details of `Undersampled-25%Labeled` can found in Table 6.2.

experiments ('All-features' and 'wheat' Table 6.2), with all others. It is also interesting to notice how little the balanced accuracy values oscillate between unbalanced and balanced sub experiments, which goes according to what was theorized in Section 6.2. This proof gives strength to our decision of using it to compare model performances between balanced and unbalanced sub experiments. To finalize what it is perceivable from these plots, we can also notice an increase of f1-score on all experiments from `dataset` to `dataset_wheat`, which partially confirms that filtering the data by plant species increases the model's performance. In fact, our models present a higher f1-score with lower amounts of the same land cover (1.218 data points in `dataset_wheat`), than larger dispersed amounts (16.131 in `dataset`).

7.2.2 SL and SSL comparison

Our second point of discussion focuses on the direct comparison between SSL and SL models on datasets with high volumes of unlabeled data. To conduct a fair and realistic assessment, we decided to

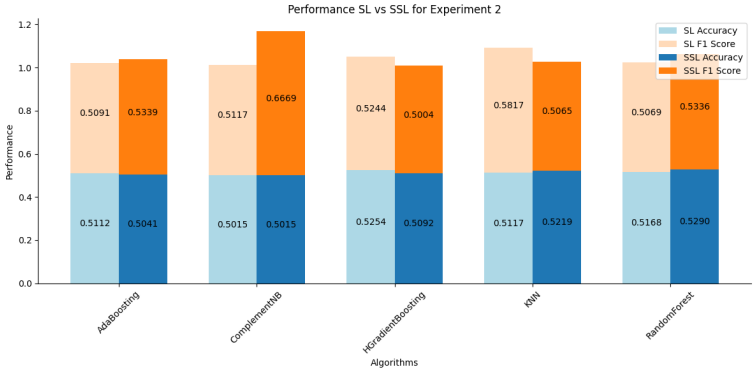


Figure 7.6: Balanced accuracy and f1-score comparison between each model and their SSL adaptation. These values only take into account the sub experiment `Undersampled-25%Labeled` of experiment 1. Note: Details of `Undersampled-25%Labeled` can found in Table 6.2.

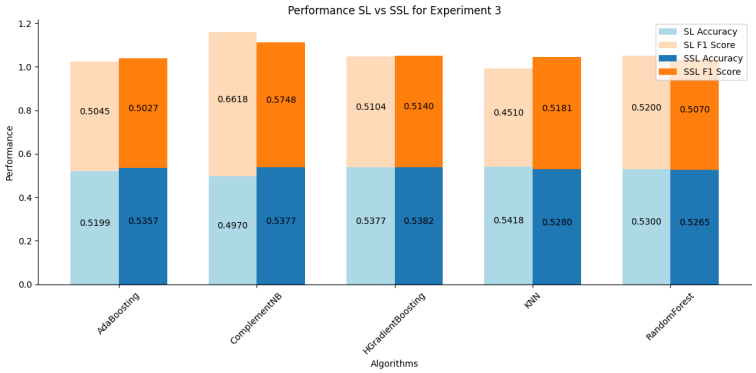


Figure 7.7: Balanced accuracy and f1-score comparison between each model and their SSL adaptation. These values only take into account the sub experiment `Undersampled-25%Labeled` of experiment 3. Note: Details of `Undersampled-25%Labeled` can found in Table 6.2.

look into the sub-experiment that had small amounts of labeled data but enough for SL to reach interesting results. This way, it provides enough resources for SL to compete with SSL, as it lays an opportunity for both to be superior. From our pool of sub-experiments, `Undersampled-25%Labeled` is the most adequate for the task at hand. On Table 6.2, we can recall its summary. Thus, three bar plots were created, (Figure 7.5, Figure 7.6 and Figure 7.7), where each represents one of the major experiments described earlier. Experiments 1, 2 and 3. Unlike the previous set of graphs, it only represents one sub-experiment (`Undersampled-25%Labeled`) but differentiates each model. Therefore, its X-axis corresponds to each of the core algorithms used. These partitions possess two bar plots, where the left one (colored with lighter tones) corresponds to the model's original architecture, whereas the one on the right represents its SSL version.

From the results obtained with Experiment 1, displayed in Figure 7.5, only two of the five models failed to demonstrate an increase in performance with their SSL adaptation. Model `AdaBoosting` maintained its overall effectiveness, and model `HGradientBoosting` decreased around 0.042. Experiment 2

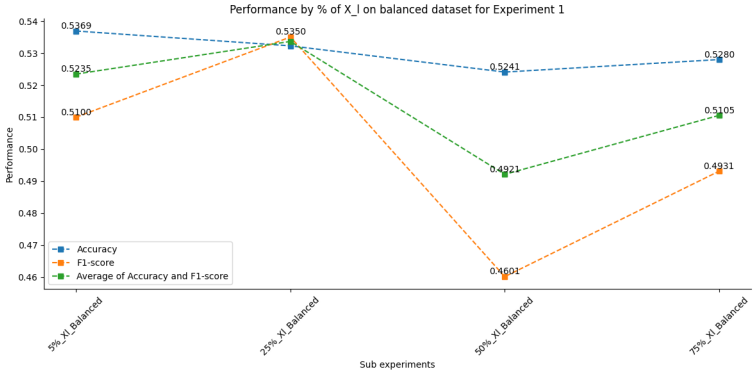


Figure 7.8: Average accuracy, average f1-score, and the average of both averages of our models results portrayed on a scatter plot by the percentage of labeled data. These results were obtained with Experiment 1.

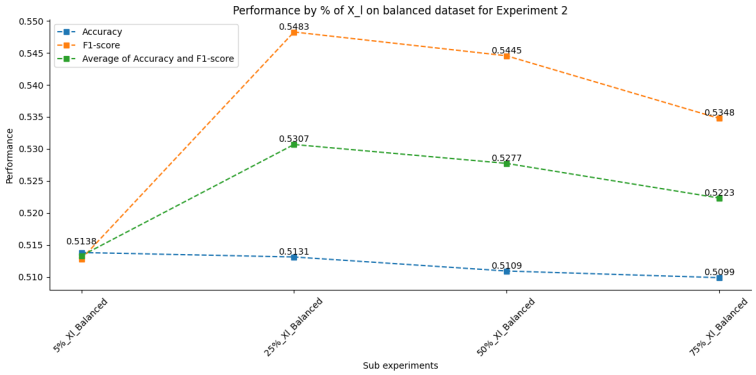


Figure 7.9: Average accuracy, average f1-score, and the average of both averages of our models results portrayed on a scatter plot by the percentage of labeled data. These results were obtained with Experiment 2.

(Figure 7.6) had similar results besides those achieved by `AdaBoosting` and `KNN`. The first one slightly increased and the second diminished around 0.06. Regarding our final experiment (Figure 7.7), only models `ComplementNB` and `RandomForest` exhibited lower performances with their SSL adaptations. So, when combining our three results, we evaluated fifteen unique comparisons, where eight had better outcomes with SSL and five with SL.

Therefore, according to this perspective, we can state that SSL holds the potential to produce slightly better outcomes more frequently than SL. Additionally, it is interesting to notice that most of our performance differences were small, except for two where SSL showed superior performance: `ComplementNB` for Experiments 1 and 2 (Figure 7.5 and Figure 7.5 respectively). These diverged in 0.259 and 0.1552, which represent higher oscillations when compared to all other. Thus, through a different perspective we reach the same conclusion: SSL performs better than their SL counterparts when 75% of data is unlabeled.

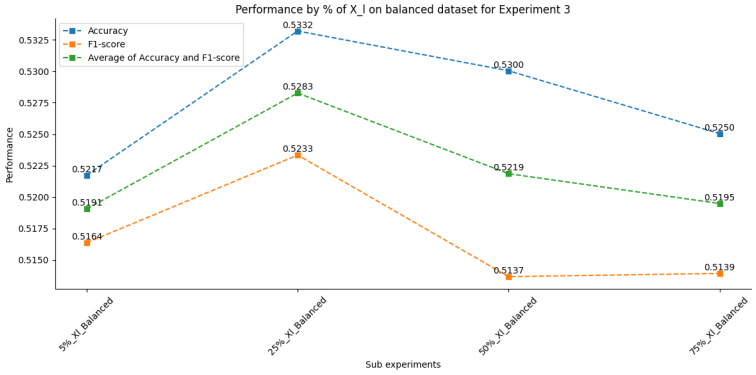


Figure 7.10: Average accuracy, average f1-score, and the average of both averages of our models results portrayed on a scatter plot by the percentage of labeled data. These results were obtained with Experiment 3.

7.2.3 Performance by percentage of X_u

To conclude our discussion, we studied the impact that the percentage of unlabeled data has on our SSL models. This behaviour was investigated by drawing the average performance reached with each of the sub experiments that suffered from undersampling and have unlabeled data. They are sub experiments `Undersampled-5%Labeled`, `Undersampled-25%Labeled`, `Undersampled-50%Labeled` and lastly `Undersampled-75%Labeled`. Thus, we created one plot per major experiment, where their X-axis corresponds to the four unique sub experiments mentioned. To obtain a general perspective, it was decided to view these results independently from our models, and so the performance levels drawn correspond to the average accuracy between all five SSL models. The same goes for the f1-score measurement. It was also included a third assessment that is calculated by averaging the average accuracy and f1-score, which allows to represent each sub experiment with one value.

Such plots can be perceived in Figure 7.8, Figure 7.9 and Figure 7.10. From them, we visualize that the three experiments reached the highest performance values when 75% of their data was unlabeled. One could say that SSL models meant to predict P values behave better when only 25% of the dataset is labeled. However, when taking a closer look into the actual values we achieved and observe that these measurements only increase around 2%, such claim cannot be taken for granted.

8

Conclusion

Contents

In this final chapter, we recall this thesis' motivations, its three main objectives, what was achieved, and ends by mentioning some obstacles, challenges, and how they can be overcome as future works.

8.1 Conclusions

Looking back into Chapter 1, this thesis had three key objectives: 1) discover or creation of an useful and high-quality dataset with soil nutrient information; 2) enhancement of the tstk python toolkit by incorporating SSL and implementations as seemed fit; and 3) utilizing this improved toolkit to study and discuss how SSL models performances compare to SL when estimating soil nutrient values through EOS data.

Concerning the dataset creation, as addressed in Chapter 4, we presented four potential datasets (AfSis, BEN, NCSS and LUCAS) that were discovered and explored to be merged as a final dataset. AfSis lacked reliance, BEN was dispersed plus had missing land cover labels, and NCSS had only a small amount of usable data. LUCAS, instead, was the only dataset that had enough potential and quality to be used as our source of data. It presented land cover and land usage, was built by a reliable organization (ESA), and had been previously applied in [6]. After a few filters and transformations, we terminated with a resourceful classification dataset. Still, since its N and K values were underrepresented, we solemnly focused on its P for the remaining experiments presented in this thesis. This dataset was named `dataset`.

Moving to our contributions to the tstk software toolkit, we ended up performing a large amount of improvements. From attribute renaming and reduction, restructuring of hierarchy, the addition of useful methods, the inclusion of SSL algorithms, and optimization of code, tstk suffered a complete restructure, which made it more clear, user-friendly, useful, and broad. These updates significantly changed its course and served as a contribution to the AI for EOS research community as a refined piece of software that allows for easy EOS data extraction and subsequent ML training and prediction.

Concerning our final objective, this study demonstrated a slight SSL superiority to classic SL models when estimating P values on datasets with high amounts of unlabeled data. Moreover, this investigation also confirmed what was theorized in the previous chapters, that performance levels increase when the training and testing data share a unique land cover (`dataset_wheat` gave better results than `dataset`), and also when balanced (all of the `Undersampled` experiments had higher performances than `dataset` core experiment). However, despite SSL demonstrating slightly superior performance, given our limited dataset and marginal performance disparities to their SL counterparts, further research is necessary to confirm our conclusions.

8.2 System Limitations, Challenges and Future Work

The first major limitation of our investigation was the inconsistency in the soil fertility literature. The papers cited in Chapter 2 disagreed on the nutrient concentrations that define fertile soil. For example, [8] indicated that soil N must be between 40 and $15,000$ ppm, and P around $2,000$ ppm (see Table 2.1). This is inconsistent with literature [43–46], which state that N values should be between 10 to 50 mg/kg, and [42] which reported optimal crop yield with P from 10.9 to 21.4 mg/kg. Similarly, for *Triticum spelta*, [48] states soil N should be between $16,300$ and $30,100$ mg/kg, and P between $2,650$ and $5,020$ mg/kg, which does not align with the general values mentioned. Furthermore, soil fertility requires more factors than EOS data, as noted in Section 2.1. Variables like pH, CEC, humidity, and bulk density influence nutrient absorption and availability, which were overlooked in our research. This omission likely contributed to inaccurate results. Predicting soil fertility, specifically P concentration, with ML proved challenging. This is seen not as a limitation, but a challenge to be addressed. Both constraints could be overcome with the addition of a soil fertility expert. With expert guidance, our future research would more accurately identify key features and methods to classify soils as fertile or infertile. Moreover, we only evaluated P due to the lack of high-quality data on other essential elements. Future research should include these nutrients. Finally, future work should focus on datasets for the same crop and geography, as soil properties and fertility requirements vary by region and crop type. Crop filtering is crucial for improved performance.

Moving to data sources, we conclude that most publicly available data fails to meet our needs. Our dataset, derived from four different sources, still lacked good representation. Thus, future work will apply DL data augmentation models to generate fake but coherent data from `dataset_wheat`. To create a high-quality hybrid dataset, we need to add more data on $Triticum\ spelta$ and train generative models on this enhanced dataset. Currently, `dataset_wheat` has too little information for DL models to achieve good results.

Another limitation is the lack of validation in tstk. Like any software, it may contain bugs that could affect our results. Given the toolkit's complexity, adding a module for unit testing is crucial to evaluate performance and rule out tstk as a source of suspicious results.

Due to the problem's complexity and our other tasks (dataset creation/validation and tstk improvement), we focused on classic ML models. Future research should use predictive DL models, which detect non-linear relationships and train on more features, overcoming our model selection limitation. Additionally, the low availability of SSL algorithms constrained our ML investigation, preventing us from discovering the latest models.

We conclude by encouraging the application of tstk to other issues. With its updated structure and accessibility, it can investigate various environmental properties using EOS data. Our contributions can be applied to land-cover or land-usage prediction, polar ice investigations, ocean research, volcanic activity, crop yield, and more. tstk is designed to handle any problem requiring EOS data [6].

Bibliography

[1] L. Zhang, L. Yang, T. Ma, F. Shen, Y. Cai, and C. Zhou, "A self-training semi-supervised machine learning method for predictive mapping of soil classes with limited sample data," *Geoderma*, vol. 384, p. 114809, 2021.

[2] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.

[3] T. Hengl, J. G. Leenaars, K. D. Shepherd, M. G. Walsh, G. Heuvelink, T. Mamo, H. Tilahun, E. Berkhout, M. Cooper, E. Fegraus *et al.*, "Soil nutrient maps of sub-saharan africa: assessment of soil nutrient content at 250 m spatial resolution using machine learning," *Nutrient Cycling in Agroecosystems*, vol. 109, no. 1, pp. 77–102, 2017.

[4] T. Hengl, M. A. Miller, J. Križan, K. D. Shepherd, A. Sila, M. Kilibarda, O. Antonijević, L. Glušica, A. Dobermann, S. M. Haefele *et al.*, "African soil properties and nutrients mapped at 30 m spatial resolution using two-scale ensemble machine learning," *Scientific Reports*, vol. 11, no. 1, pp. 1–18, 2021.

[5] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *arXiv preprint arXiv:2006.05278*, 2020.

[6] Manuel Pereira, "TerraSenseTK: A Toolkit for Remote Soil Nutrient Estimation."

[7] L.-H. Jia, L.-Z. Guo, Z. Zhou, and Y.-F. Li, "Lamda-ssl: Semi-supervised learning in python," *arXiv preprint arXiv:2208.04610*, 2022.

[8] J. L. Havlin, "Soil: Fertility and nutrient management," in *Landscape and land capacity*. CRC Press, 2020, pp. 251–265.

[9] K. Mengel, "Potassium," in *Handbook of plant nutrition*. CRC press, 2016, pp. 107–136.

[10] H. Van Es, "A new definition of soil," *Csa News*, vol. 62, no. 10, pp. 20–21, 2017.

[11] N. Alexandratos and J. Bruinsma, "World agriculture towards 2030/2050: the 2012 revision," 2012.

[12] J. Dewis, F. Freitas *et al.*, "Physical and chemical methods of soil and water analysis." *FAO soils Bulletin*, no. 10, 1970.

[13] E. K. Bünemann, G. Bongiorno, Z. Bai, R. E. Creamer, G. De Deyn, R. de Goede, L. Fleskens, V. Geissen, T. W. Kuyper, P. Mäder *et al.*, "Soil quality–a critical review," *Soil Biology and Biochemistry*, vol. 120, pp. 105–125, 2018.

[14] J. Trontelj ml and O. Chambers, "Machine learning strategy for soil nutrients prediction using spectroscopic method," *Sensors*, vol. 21, no. 12, p. 4208, 2021.

[15] F. A. Diaz-Gonzalez, J. Vuelvas, C. A. Correa, V. E. Vallejo, and D. Patino, "Machine learning and remote sensing techniques applied to estimate soil indicators – review," *Ecological Indicators*, vol. 135, p. 108517, 2022.

[16] J. Padarian, B. Minasny, and A. B. McBratney, "Machine learning and soil sciences: A review aided by machine learning tools," *Soil*, vol. 6, no. 1, pp. 35–52, 2020.

[17] S. Motia and S. Reddy, "Exploration of machine learning methods for prediction and assessment of soil properties for agricultural soil management: a quantitative evaluation," in *Journal of Physics: Conference Series*, vol. 1950, no. 1. IOP Publishing, 2021, p. 012037.

[18] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[19] I. Pôças, A. Calera, I. Campos, and M. Cunha, "Remote sensing for estimating and mapping single and basal crop coefficientes: A review on spectral vegetation indices approaches," *Agricultural Water Management*, vol. 233, p. 106081, 2020.

[20] A. Bannari, D. Morin, F. Bonn, and A. Huete, "A review of vegetation indices," *Remote sensing reviews*, vol. 13, no. 1-2, pp. 95–120, 1995.

[21] M. E. Sumner and W. P. Miller, "Cation exchange capacity and exchange coefficients," *Methods of soil analysis: Part 3 Chemical methods*, vol. 5, pp. 1201–1229, 1996.

[22] S. C. Hodges, "Soil fertility basics," *Soil Science Extension, North Carolina State University*, 2010.

[23] P. Panagos, M. Van Liedekerke, P. Borrelli, J. Köninger, C. Ballabio, A. Orgiazzi, E. Lugato, L. Liakos, J. Hervas, A. Jones *et al.*, "European soil data centre 2.0: Soil data and knowledge in support of the eu policies," *European Journal of Soil Science*, vol. 73, no. 6, p. e13315, 2022.

[24] M. Dedeoğlu, L. Başayiğit, M. Yüksel, and F. Kaya, "Assessment of the vegetation indices on sentinel-2a images for predicting the soil productivity potential in bursa, turkey," *Environmental Monitoring and Assessment*, vol. 192, no. 1, pp. 1–16, 2020.

[25] A. Gholamy, V. Kreinovich, and O. Kosheleva, “Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation,” 2018.

[26] F. Naumann, “Data profiling revisited,” *ACM SIGMOD Record*, vol. 42, no. 4, pp. 40–49, 2014.

[27] V. Barnett, T. Lewis *et al.*, *Outliers in statistical data*. Wiley New York, 1994, vol. 3, no. 1.

[28] G. Sumbul, “Bigearthnet-mm: A large scale multi-modal multi-label benchmark archive for remote sensing image classification and retrieval,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 9, no. 3, pp. 174–180, 2021.

[29] G. Sumbul, M. Charfuelan, B. Demir, and V. Markl, “Bigearthnet: A large-scale benchmark archive for remote sensing image understanding,” in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2019, pp. 5901–5904.

[30] T. Reinsch, L. West *et al.*, “The us national cooperative soil characterization database,” in *Proceeding of the 19th World Congress of Soil Science*, 2010, pp. 1–6.

[31] A. Ramcharan, T. Hengl, D. Beaudette, and S. Wills, “A soil bulk density pedotransfer function based on machine learning: A case study with the ncss soil characterization database,” *Soil Science Society of America Journal*, vol. 81, no. 6, pp. 1279–1287, 2017.

[32] A. Orgiazzi, C. Ballabio, P. Panagos, A. Jones, and O. Fernández-Ugalde, “Lucas soil, the largest expandable soil dataset for europe: a review,” *European Journal of Soil Science*, vol. 69, no. 1, pp. 140–153, 2018.

[33] R. d’Andrimont, A. Verhegghen, M. Meroni, G. Lemoine, P. Strobl, B. Eiselt, M. Yordanov, L. Martinez-Sanchez, and M. van der Velde, “Lucas copernicus 2018: Earth-observation-relevant in situ data on land cover and use throughout the european union,” *Earth System Science Data*, vol. 13, no. 3, pp. 1119–1133, 2021.

[34] R. Board and L. Pitt, “Semi-supervised learning,” *Machine Learning*, vol. 4, pp. 41–65, 1989.

[35] D. Yarowsky, “Unsupervised word sense disambiguation rivaling supervised methods,” in *33rd annual meeting of the association for computational linguistics*, 1995, pp. 189–196.

[36] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[37] I. Triguero, S. García, and F. Herrera, “Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study,” *Knowledge and Information systems*, vol. 42, pp. 245–284, 2015.

[38] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[39] W. Han, R. Feng, L. Wang, and Y. Cheng, "A semi-supervised generative framework with deep learning features for high-resolution remote sensing image scene classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 23–43, 2018.

[40] F. Staccone, "Deep learning for sea-ice classification on synthetic aperture radar (sar) images in earth observation. classification using semi-supervised generative adversarial networks on partially labeled data," 2020.

[41] Y. Wang, H. Chen, Y. Fan, W. Sun, R. Tao, W. Hou, R. Wang, L. Yang, Z. Zhou, L.-Z. Guo, H. Qi, Z. Wu, Y.-F. Li, S. Nakamura, W. Ye, M. Savvides, B. Raj, T. Shinozaki, B. Schiele, J. Wang, X. Xie, and Y. Zhang, "Usb: A unified semi-supervised learning benchmark for classification," 2022. [Online]. Available: https://arxiv.org/abs/2208.07204

[42] Z. Bai, H. Li, X. Yang, B. Zhou, X. Shi, B. Wang, D. Li, J. Shen, Q. Chen, W. Qin *et al.*, "The critical soil p levels for crop yield, soil fertility and environmental safety in different soil types," *Plant and Soil*, vol. 372, pp. 27–37, 2013.

[43] A. B. Pattison, P. Moody, and J. Bagshaw, "Vegetable plant and soil health," 2010.

[44] D. Geisseler and W. R. Horwath, "Sampling for soil nitrate determination," 2016.

[45] G. Griffin, W. Jokela, D. Ross, D. Pettinelli, T. Morris, and A. Wilf, "Recommended soil nitrate-n tests," 1995.

[46] J. D. R. Thomas, *Ion-Selective Electrode Reviews: Volume 7*. Elsevier, 2017, vol. 7.

[47] W. Kreuser, "Simplifying soil test interpretations for turf professionals," *G2265: http://turf. unl. edu/NebGuides/g2265. pdf (accessed March 3, 2017)*, 2015.

[48] N. Huang, H. Dang, W. Mu, J. Ma, Y. Ma, L. Wang, M. Shi, H. Tian, J. Liu, Y. Chen *et al.*, "High yield with efficient nutrient use: Opportunities and challenges for wheat," *Iscience*, 2023.

[49] 2017. [Online]. Available: https://eo-learn.readthedocs.io/en/latest/examples/core/CoreOverview.html#EOPatch

[50] A. J. Richardson and J. H. Everitt, "Using spectral vegetation indices to estimate rangeland productivity," *Geocarto International*, vol. 7, no. 1, pp. 63–69, 1992.

[51] K. Kawamura, A. D. Mackay, M. P. Tuohy, K. Betteridge, I. D. Sanches, and Y. Inoue, "Potential for spectral indices to remotely sense phosphorus and potassium content of legume-based pasture

as a means of assessing soil phosphorus and potassium fertility status," *International Journal of Remote Sensing*, vol. 32, no. 1, pp. 103–124.

[52] C. Lin, R. Ma, Q. Zhu, and J. Li, "Using hyper-spectral indices to detect soil phosphorus concentration for various land use patterns," *Environmental monitoring and assessment*, vol. 187, pp. 1–10, 2015.

[53] Y. Özyiğit and M. a. Bilgen, "Use of spectral reflectance values for determining nitrogen, phosphorus, and potassium contents of rangeland plants," *Journal of Agricultural Science and Technology*, vol. 15, no. 7, 2013. [Online]. Available: http://jast.modares.ac.ir/article-23-5138-en.html

[54] M. Maleki, L. Van Holm, H. Ramon, R. Merckx, J. De Baerdemaeker, and A. Mouazen, "Phosphorus sensing for fresh soils using visible and near infrared spectroscopy," *Biosystems Engineering*, vol. 95, no. 3, pp. 425–436, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511006002704

Important

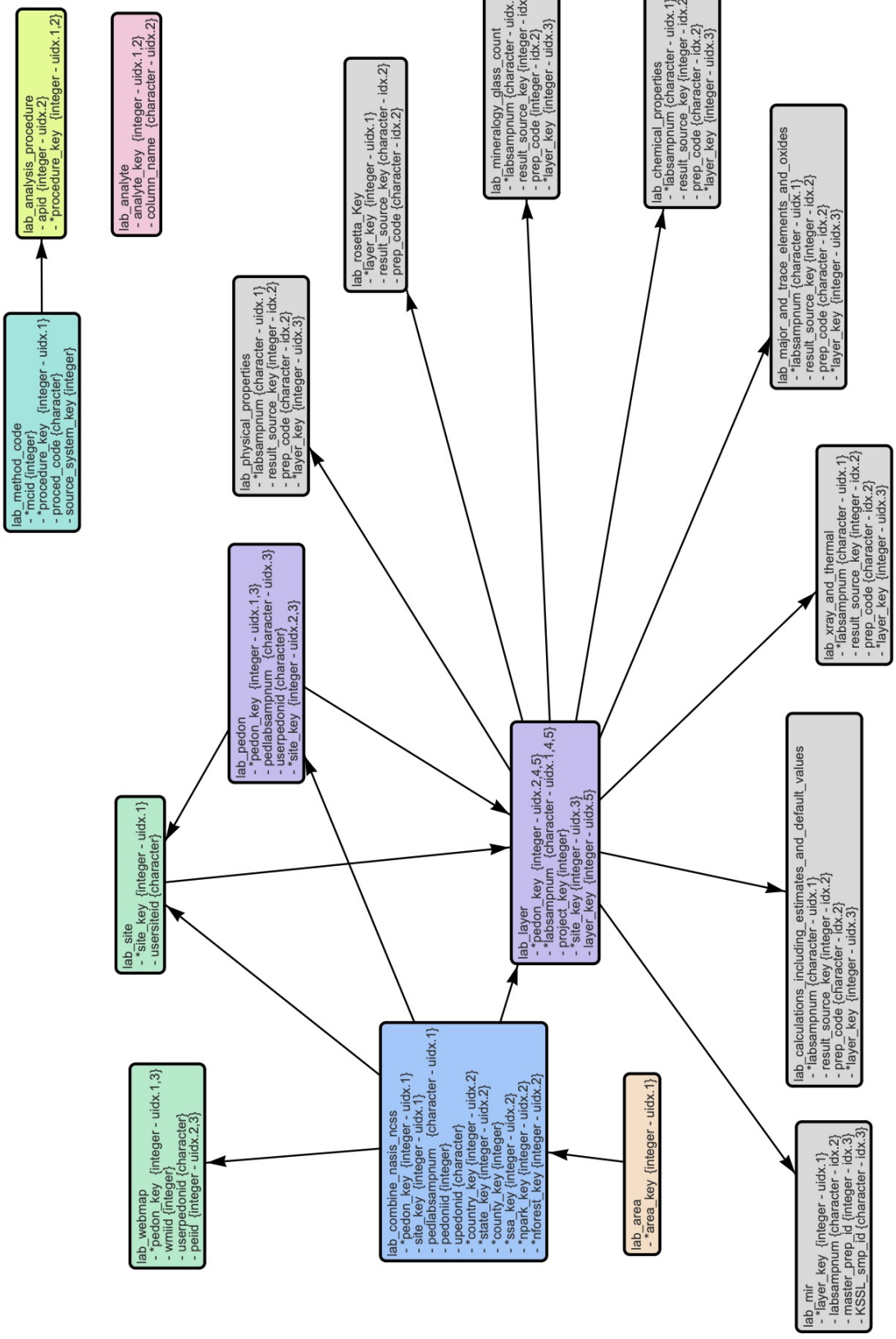


Figure A.1: NCSS original database structure.

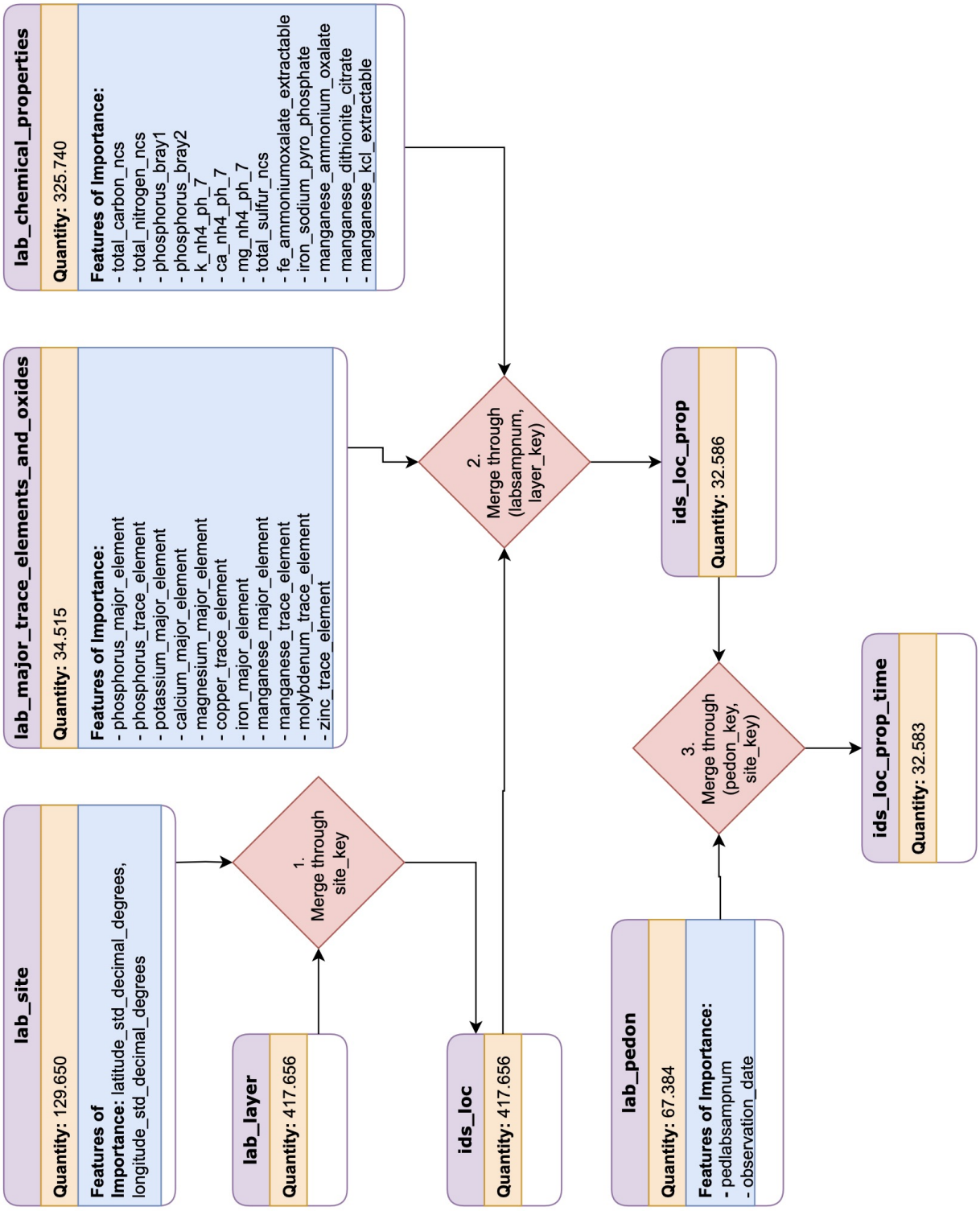


Figure A.2: NCSS diagram showing the process of converting multi table to single table.

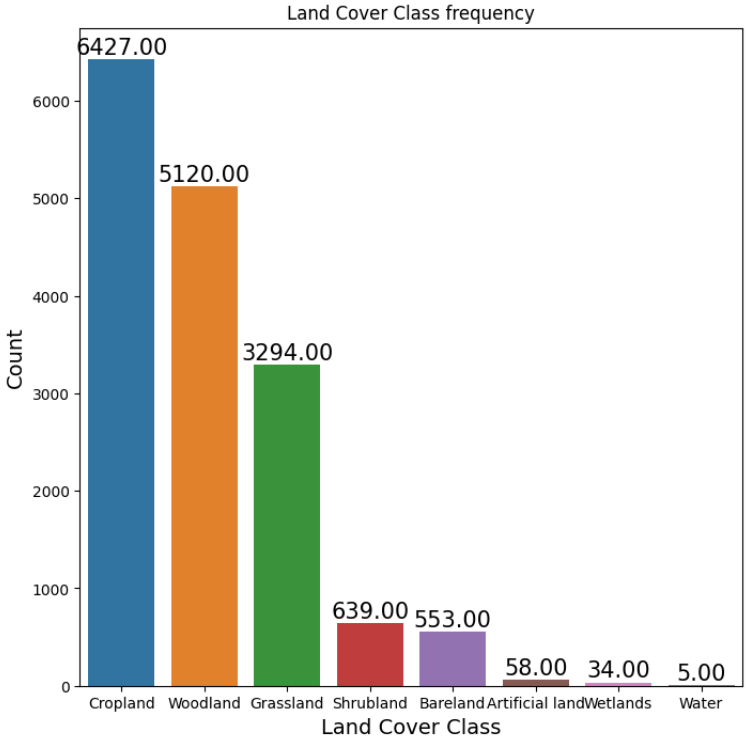


Figure A.3: Land Cover 0 frequency.

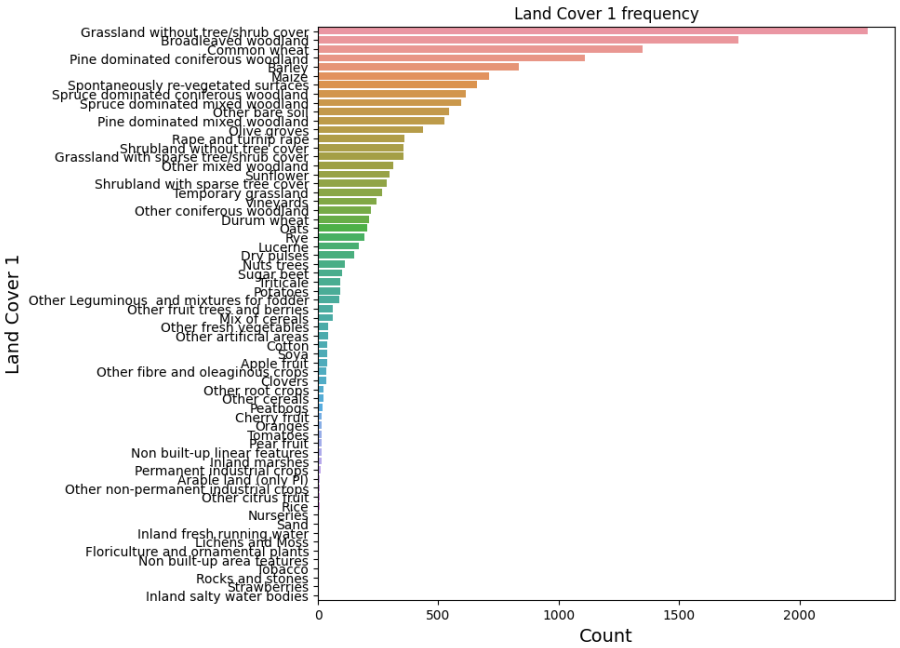


Figure A.4: Land Cover 1 frequency.

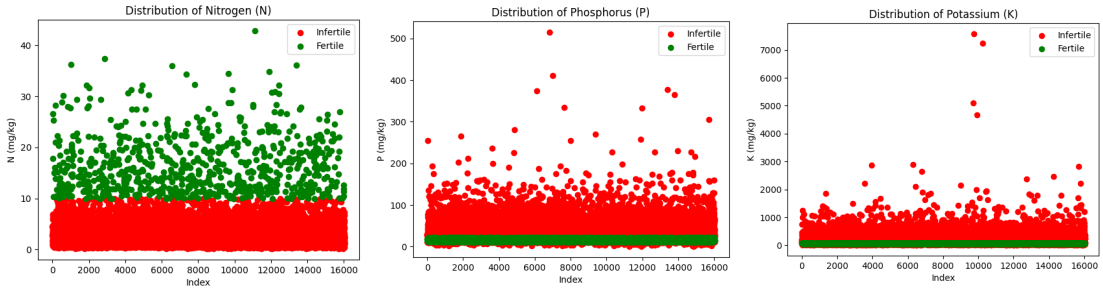


Figure A.5: N, P and K distribution in final dataset following global fertility. All values inside their fertility limits are colored in green.

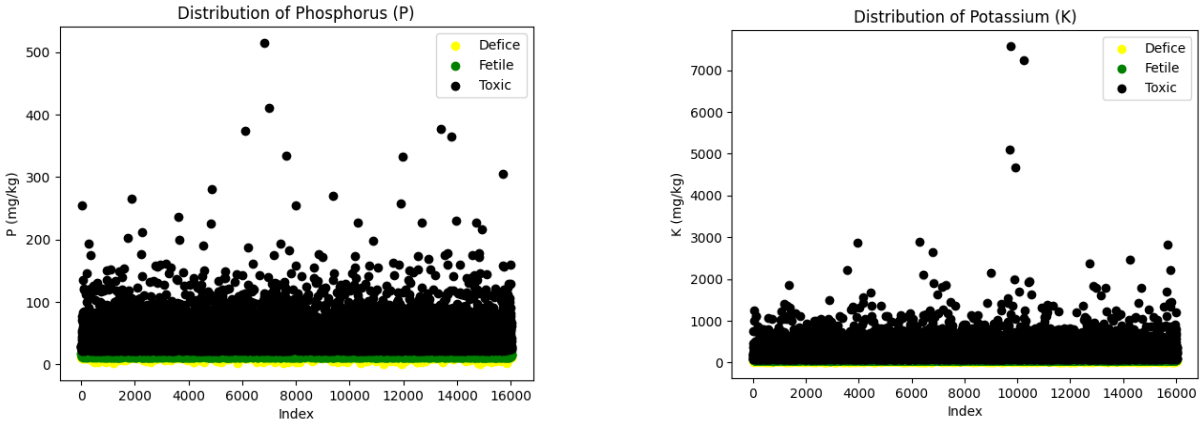


Figure A.6: P and K distribution in final dataset following global fertility. All values inside their fertility limits are colored in green. Data points bellow these limits are in yellow, and the remaining in black.

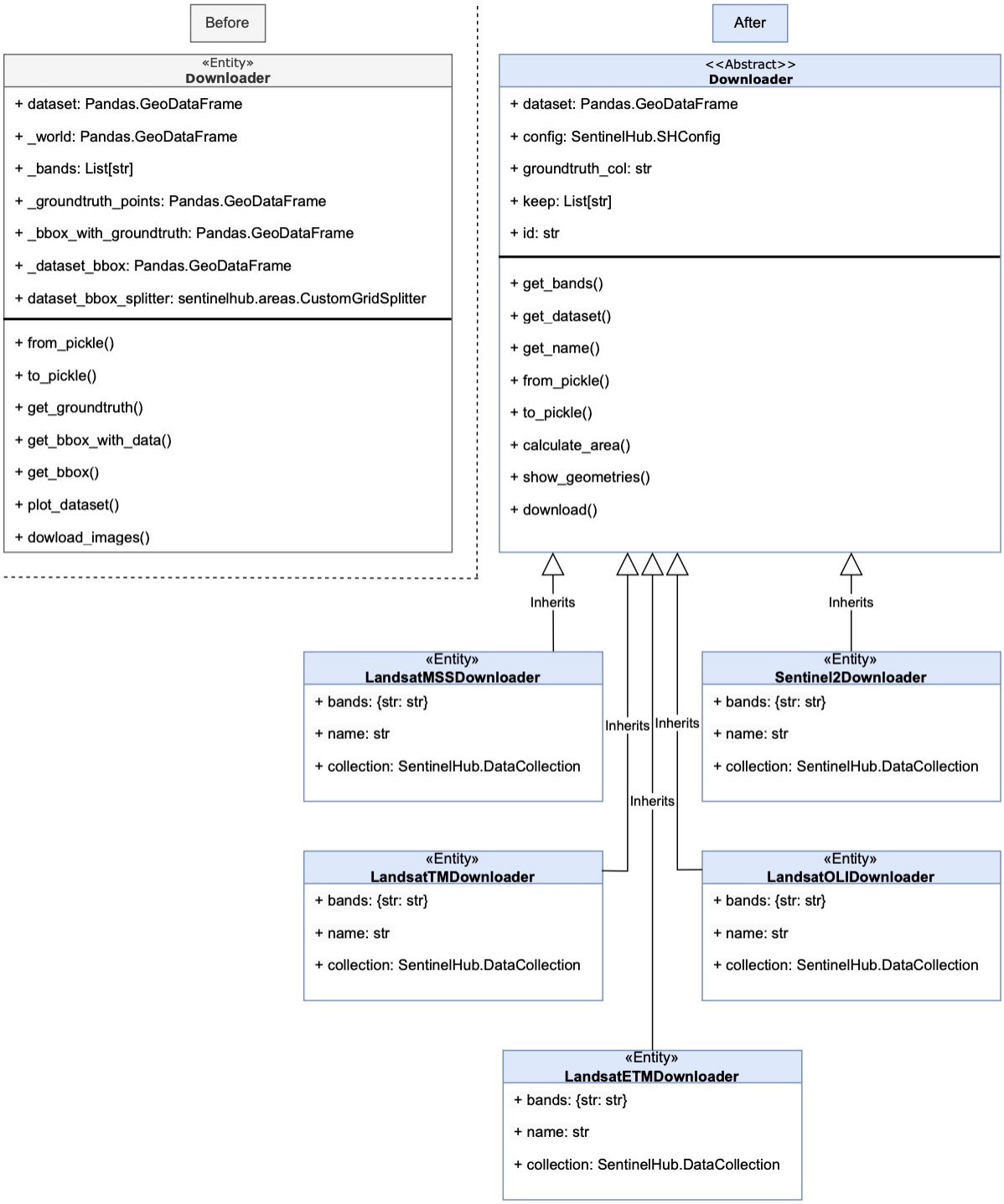


Figure A.7: Downloader class and organization before and after restructure.

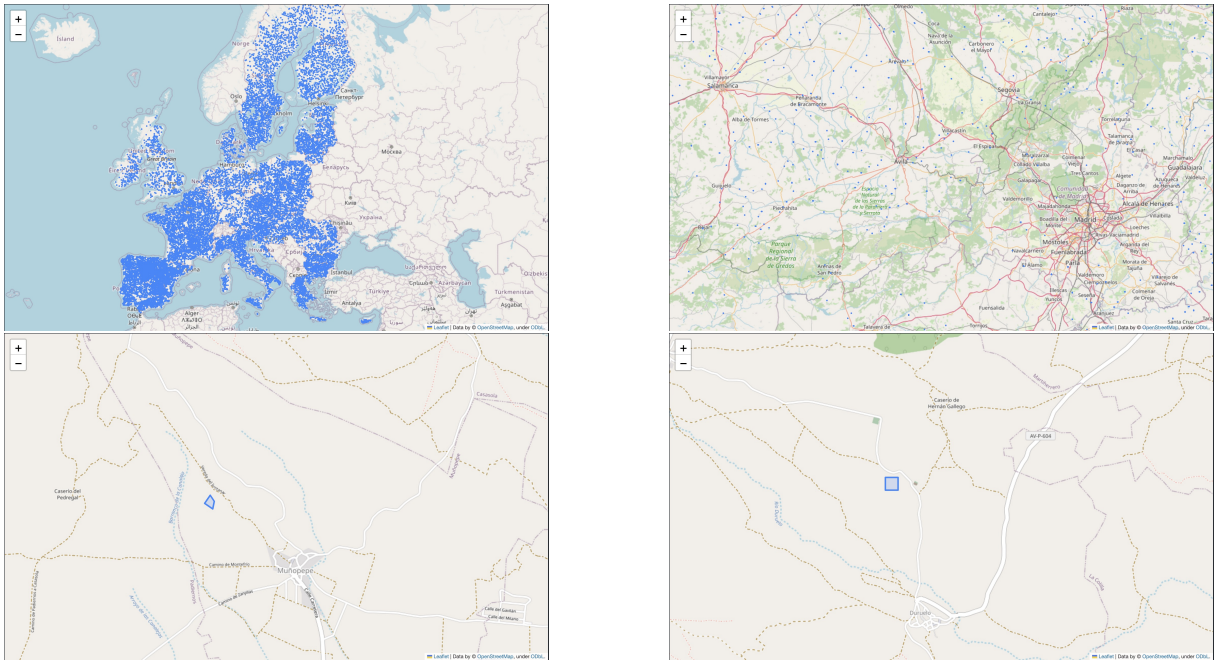


Figure A.8: Maps displayed from executing $Sentinel2Downloader.show\_geometries()$ with the Final Dataset used as shapefile. Each blue polygon represents an area of EOS data extraction. From left to right: All areas present in the final dataset; Final dataset zoomed in in Spain; Zooming in a $shapely.Polygon()$ that came with the final dataset; Location with missing area data and so transformed to $60m^2$ square centered in the $shapely.Point()$ obtained from its latitude and longitude. Notice the different shape between the measured LUCAS Copernicus area in figure 3, and the unusual perfect square calculated from $shapely.buffer()$ function.

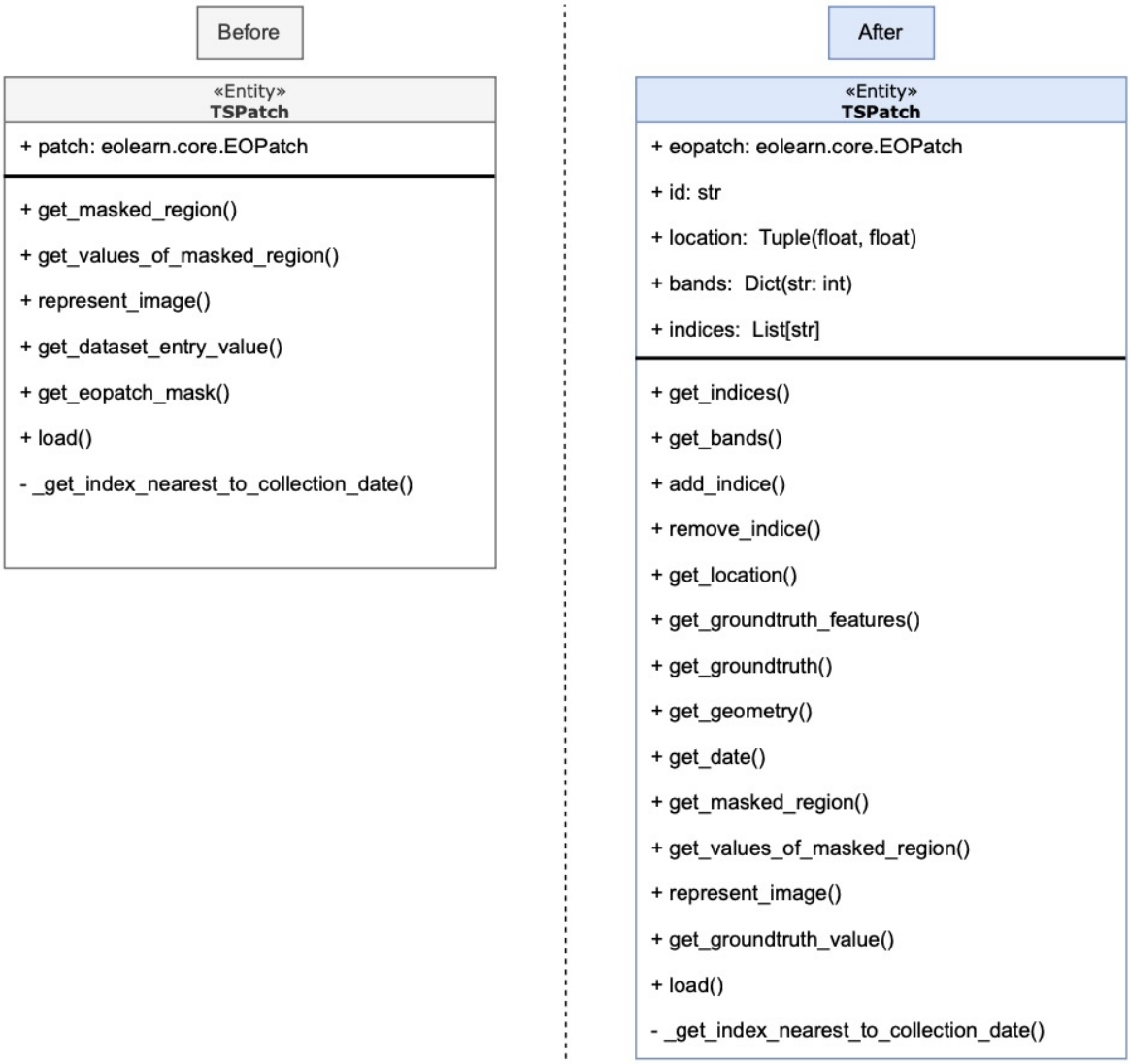


Figure A.9: `TSPatch` class before and after restructure.

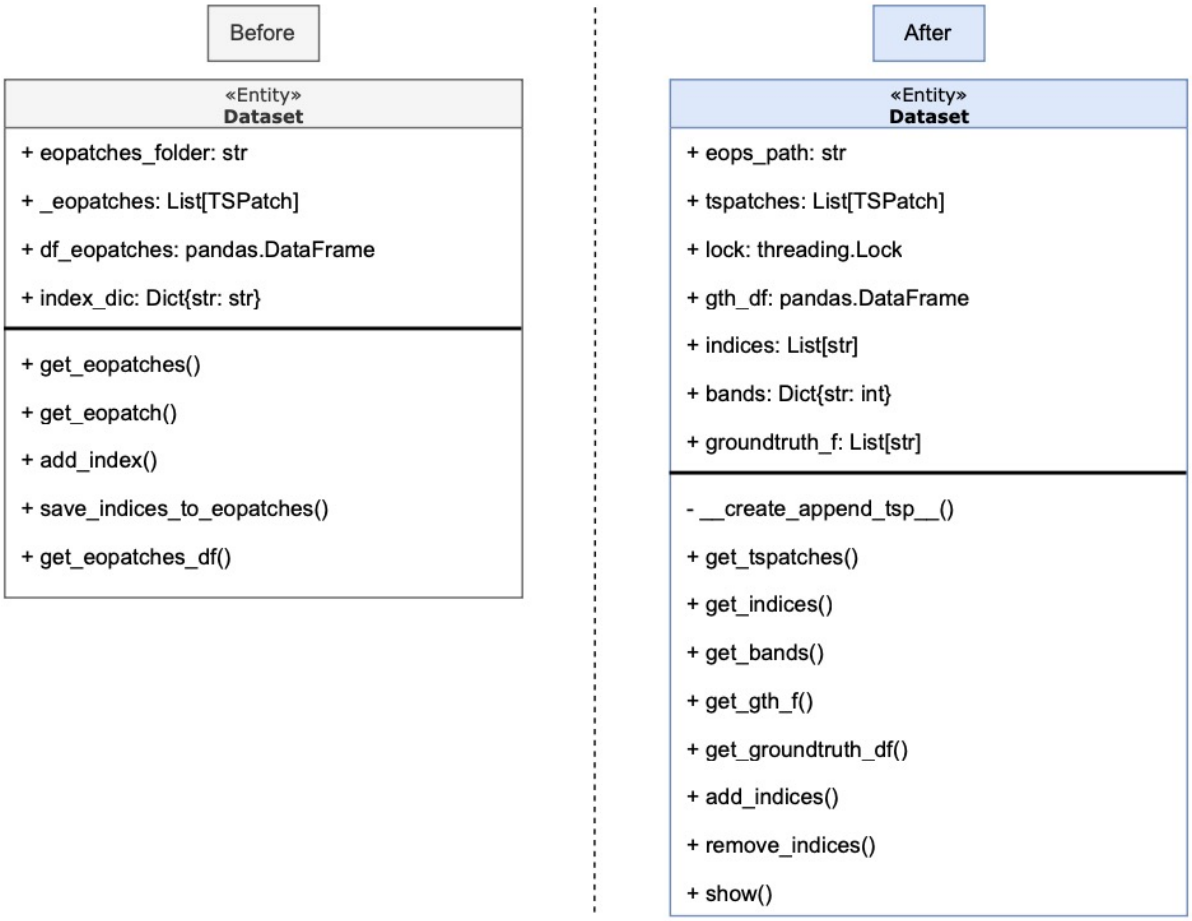


Figure A.10: `Dataset` class before and after restructuring.

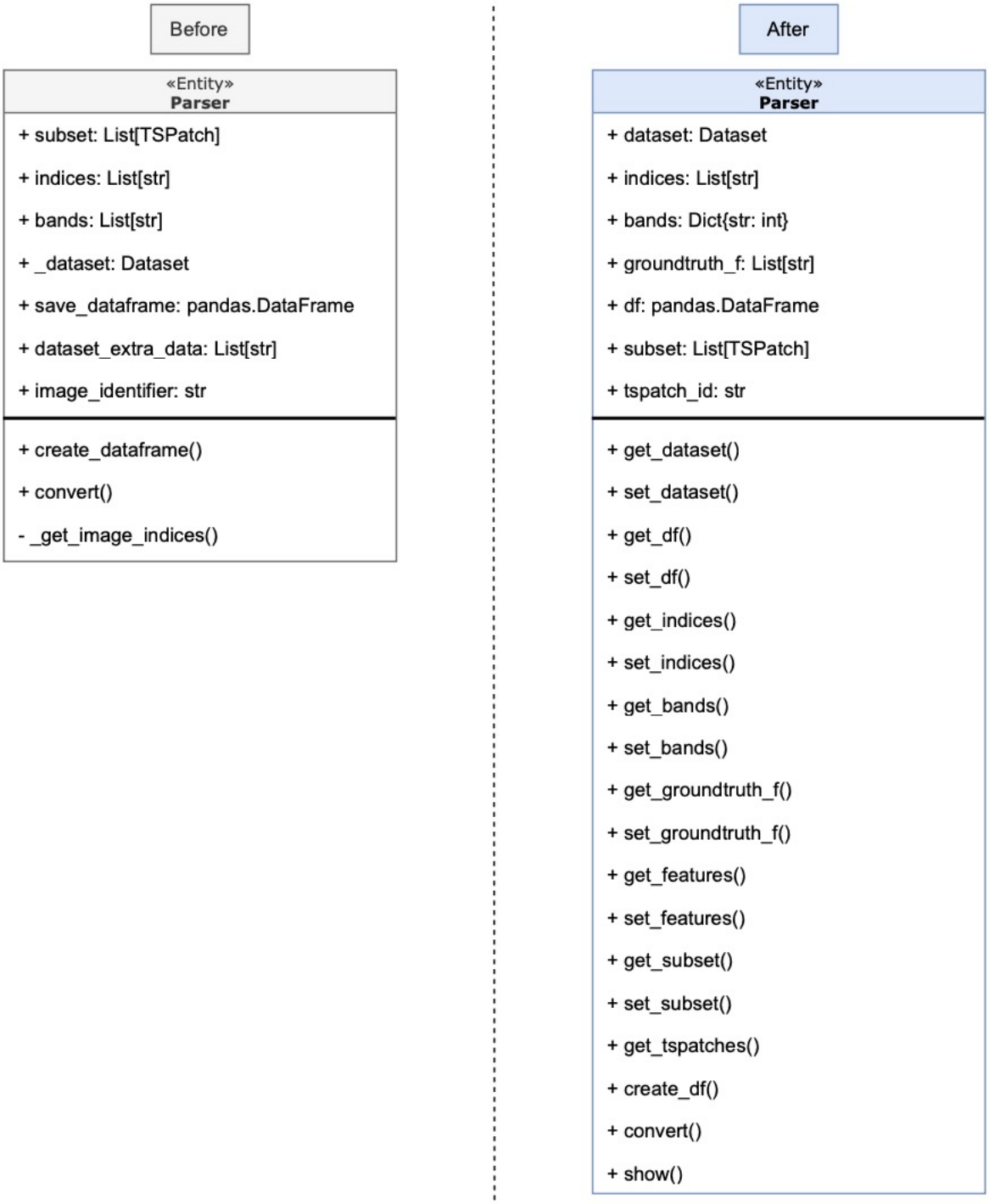


Figure A.11: `Parser` class before and after restructuring.

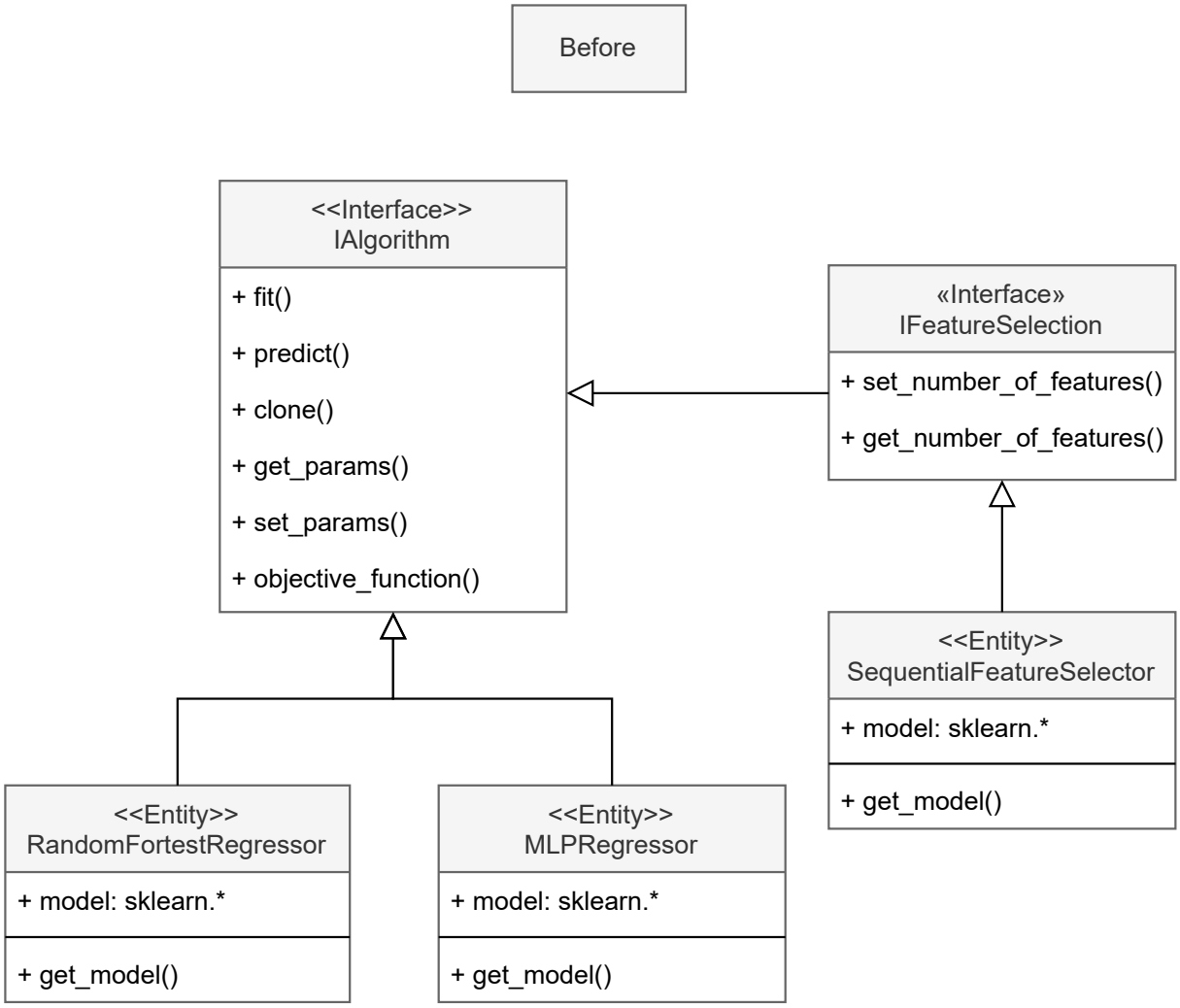


Figure A.12: `Algorithms` module before restructuring.

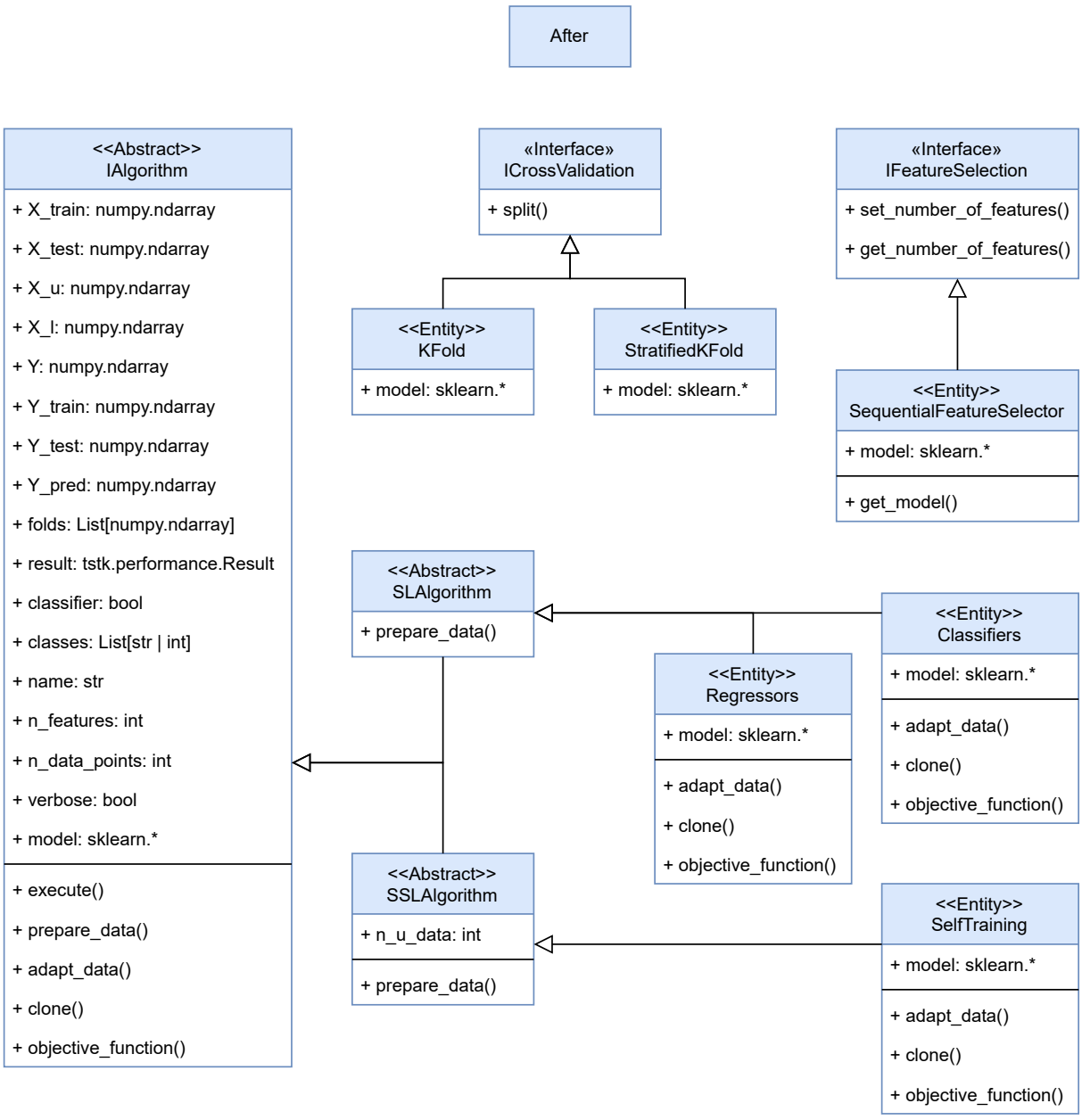


Figure A.13: `Algorithms` module after restructuring. (`Classifiers` and `Regressors` represent several implemented models of each type, and are not a real class in the toolkit)

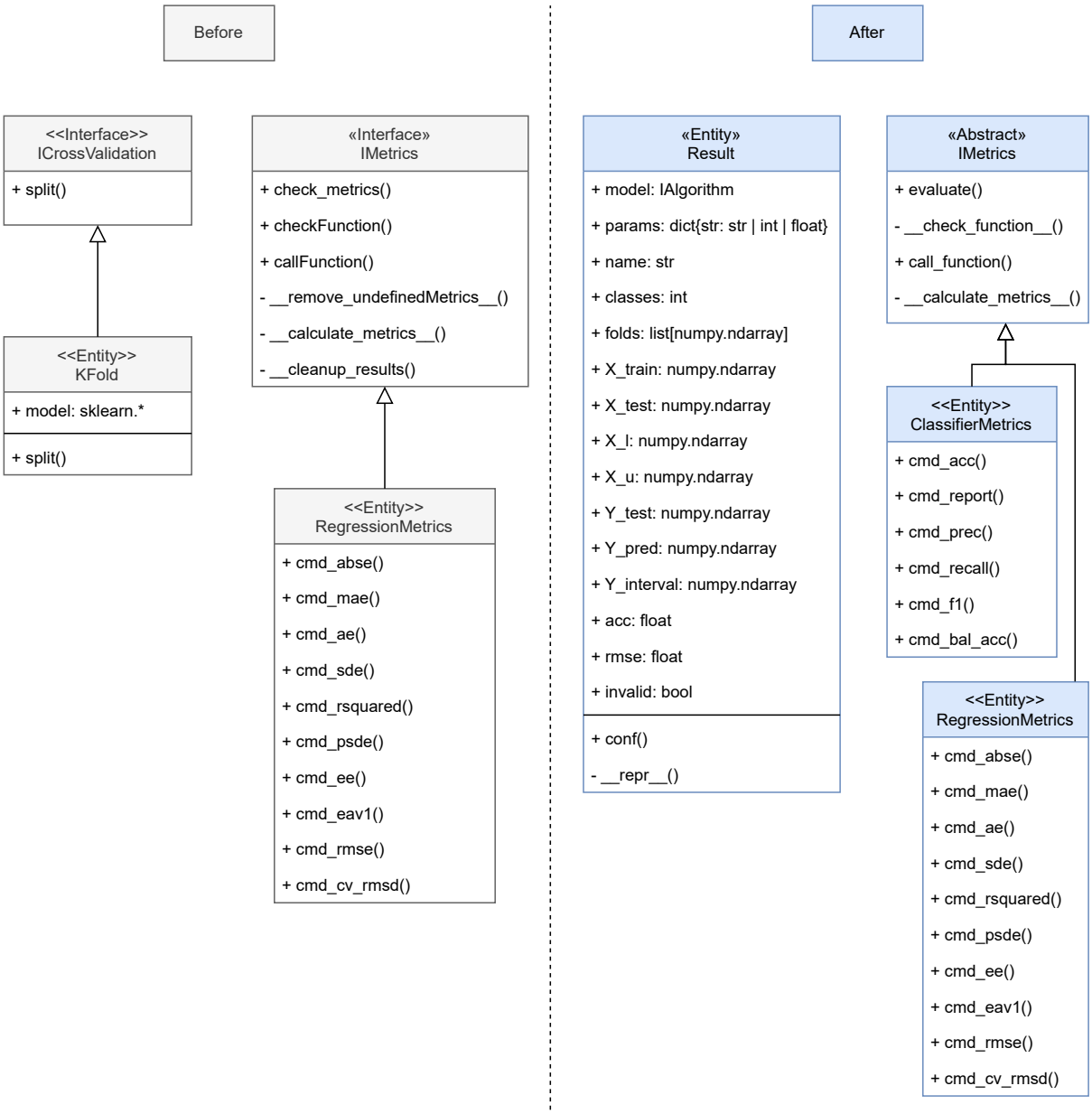


Figure A.14: `Performance` module after restructuring.

Figure A.15: `Experiment` module after restructuring.

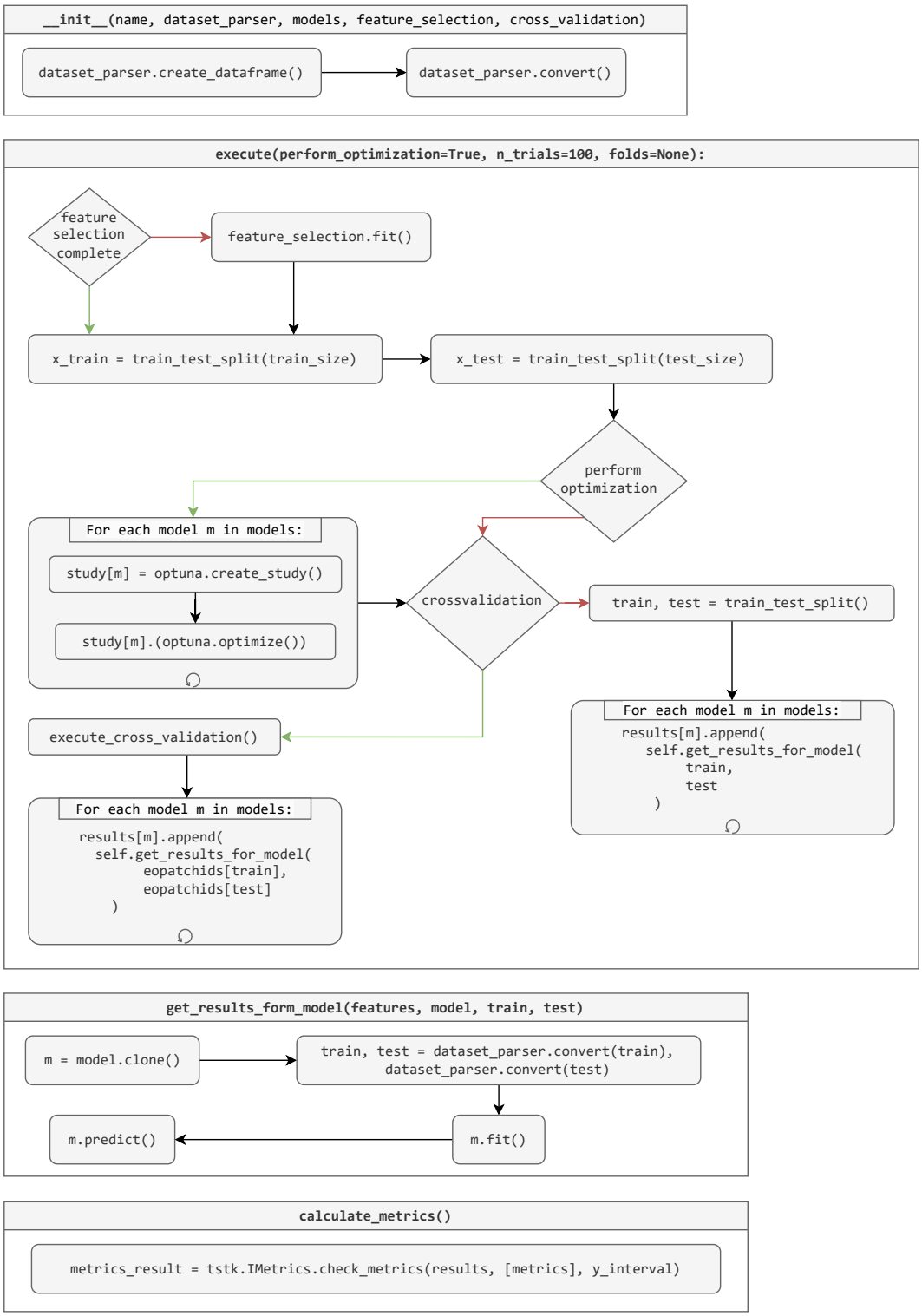


Figure A.16: `Experiment` flow graph before restructuring.

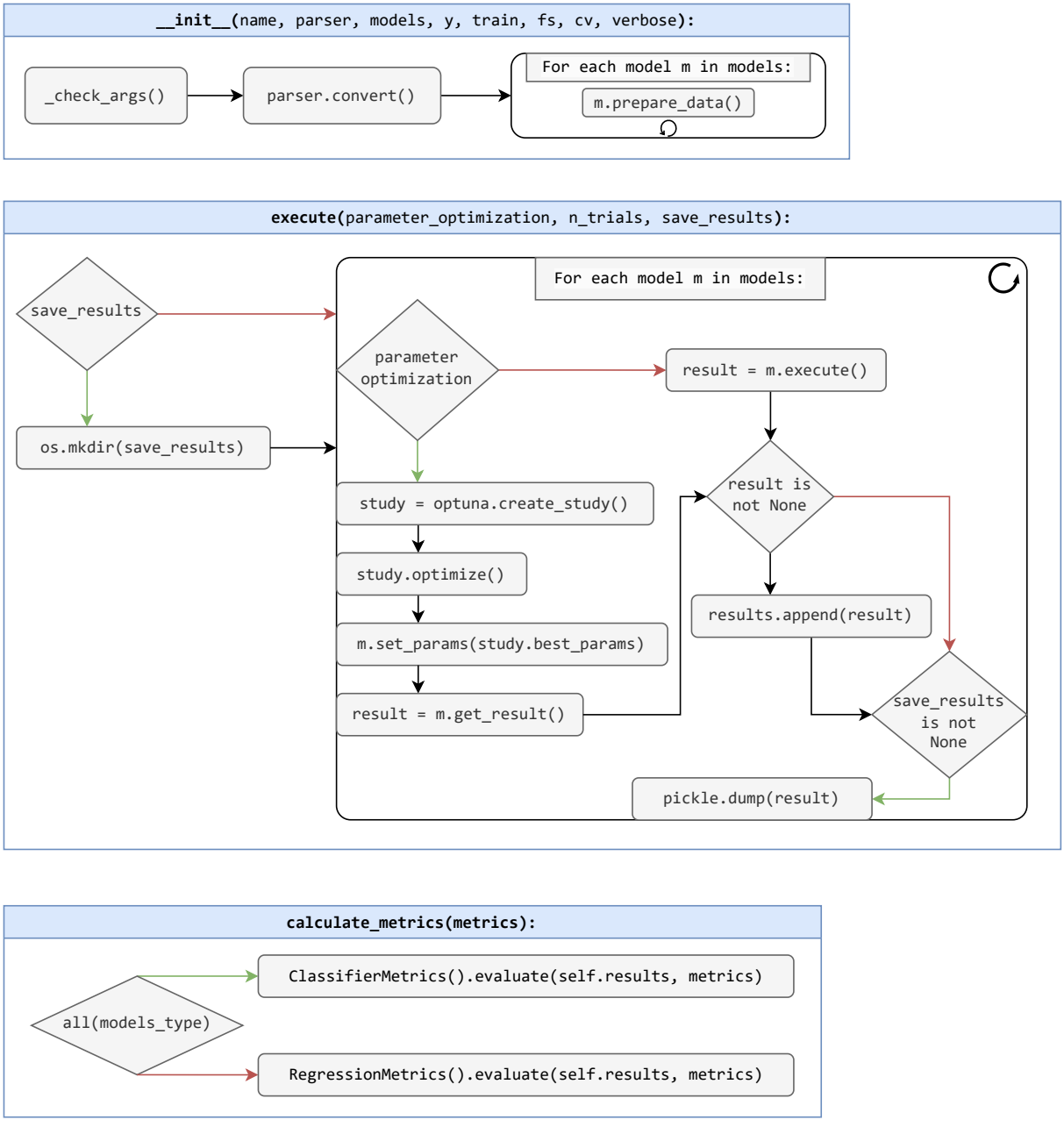


Figure A.17: `Experiment` flow graph after restructuring.

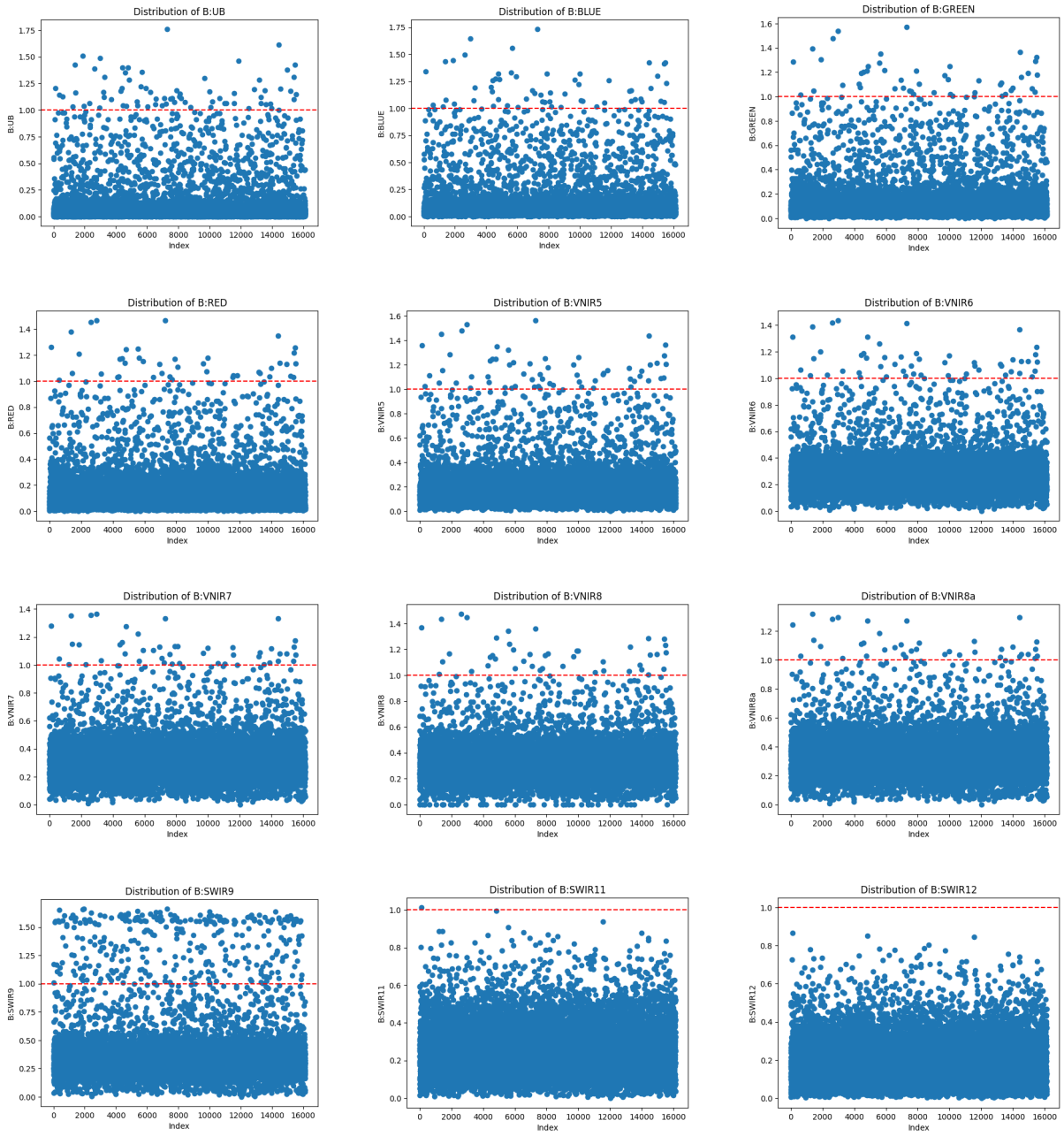


Figure A.18: Band distribution across `dataset`, which should not be higher than 1.0 or lower than 0.0. Note: Red lines correspond to these limitations.

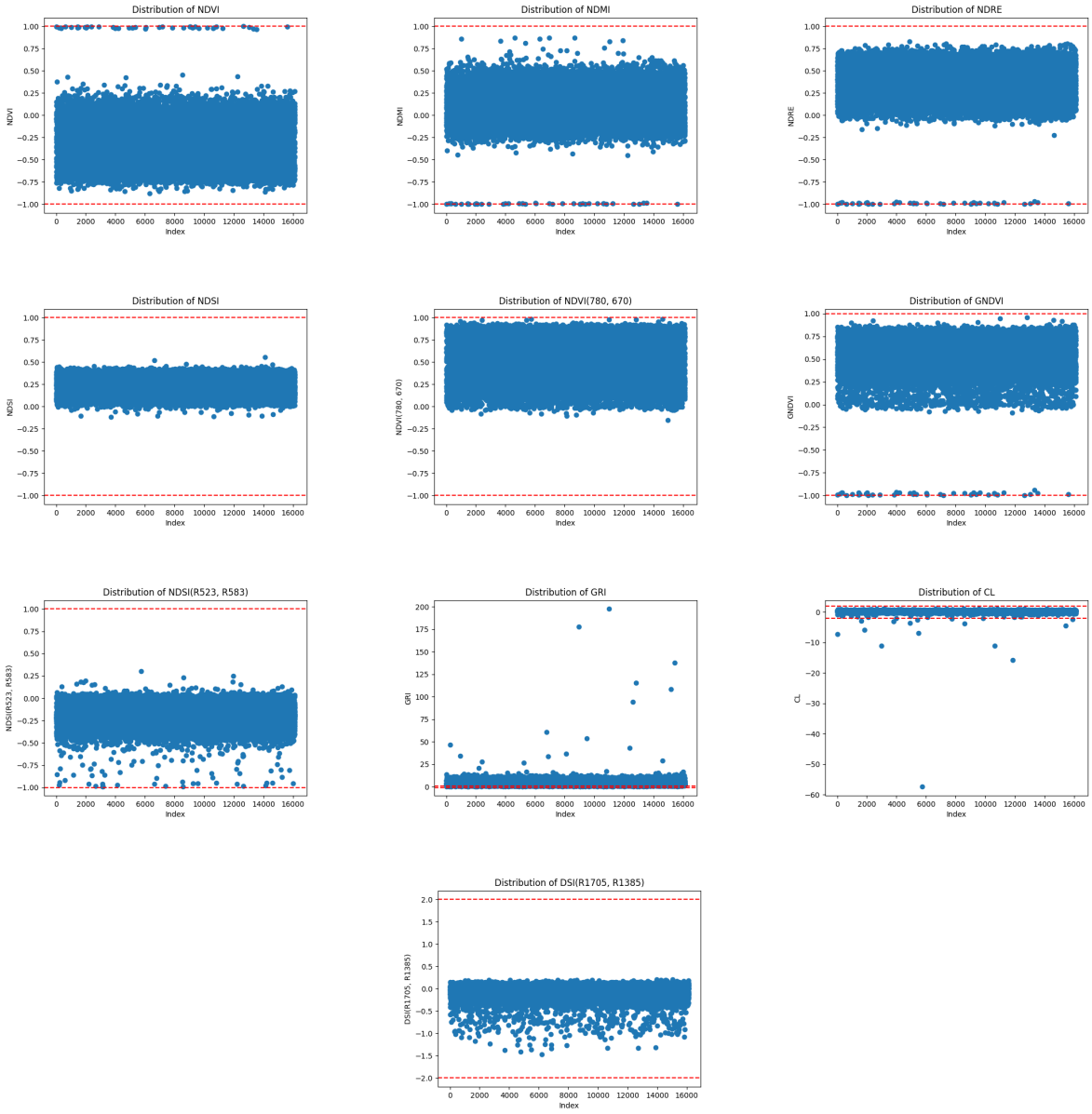


Figure A.19: SVI distribution across `dataset`. Note: Red lines correspond to the data range. More SVI were considered, but only these had range limitations.

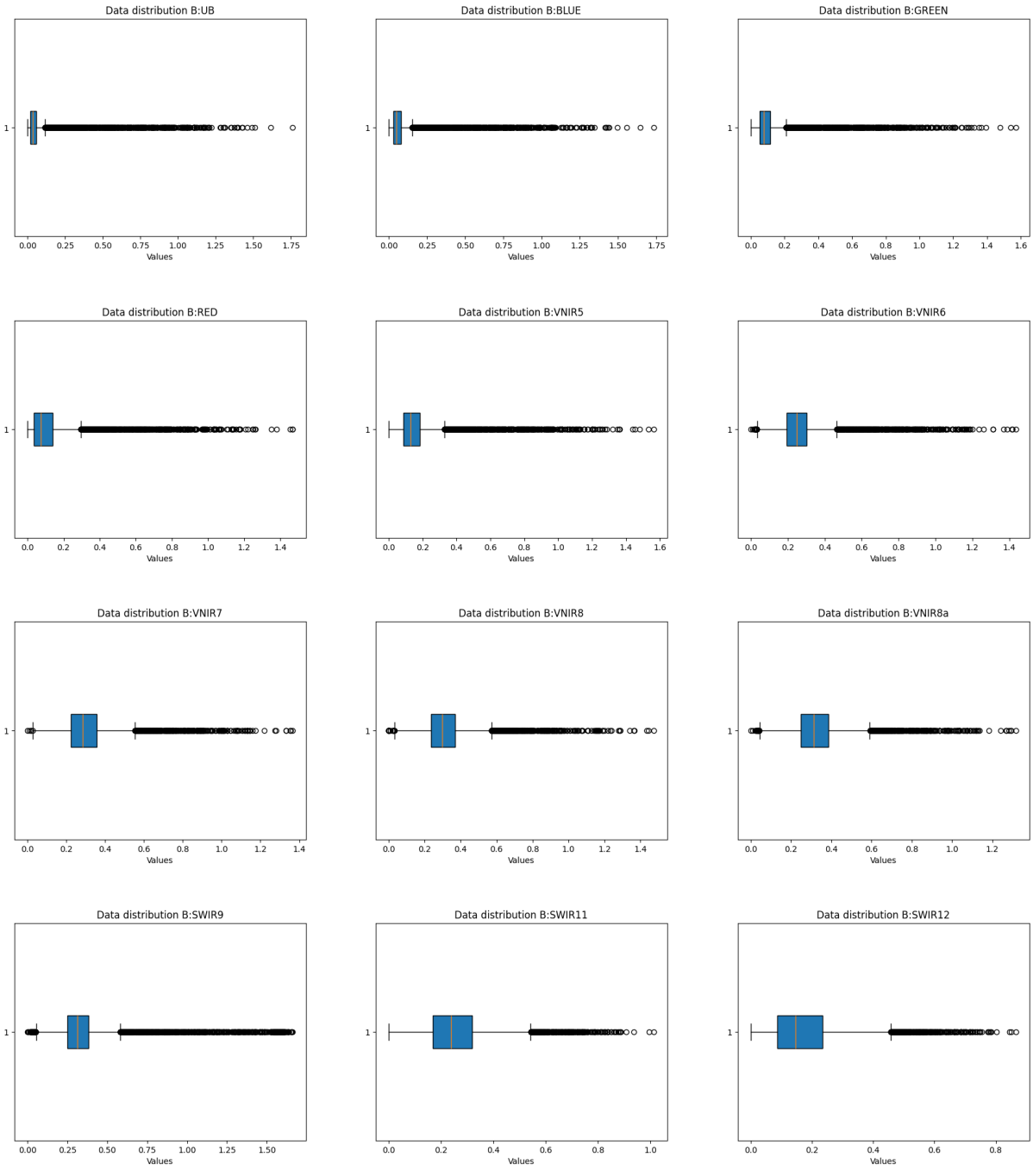


Figure A.20: Bands box-plots from `dataset`.

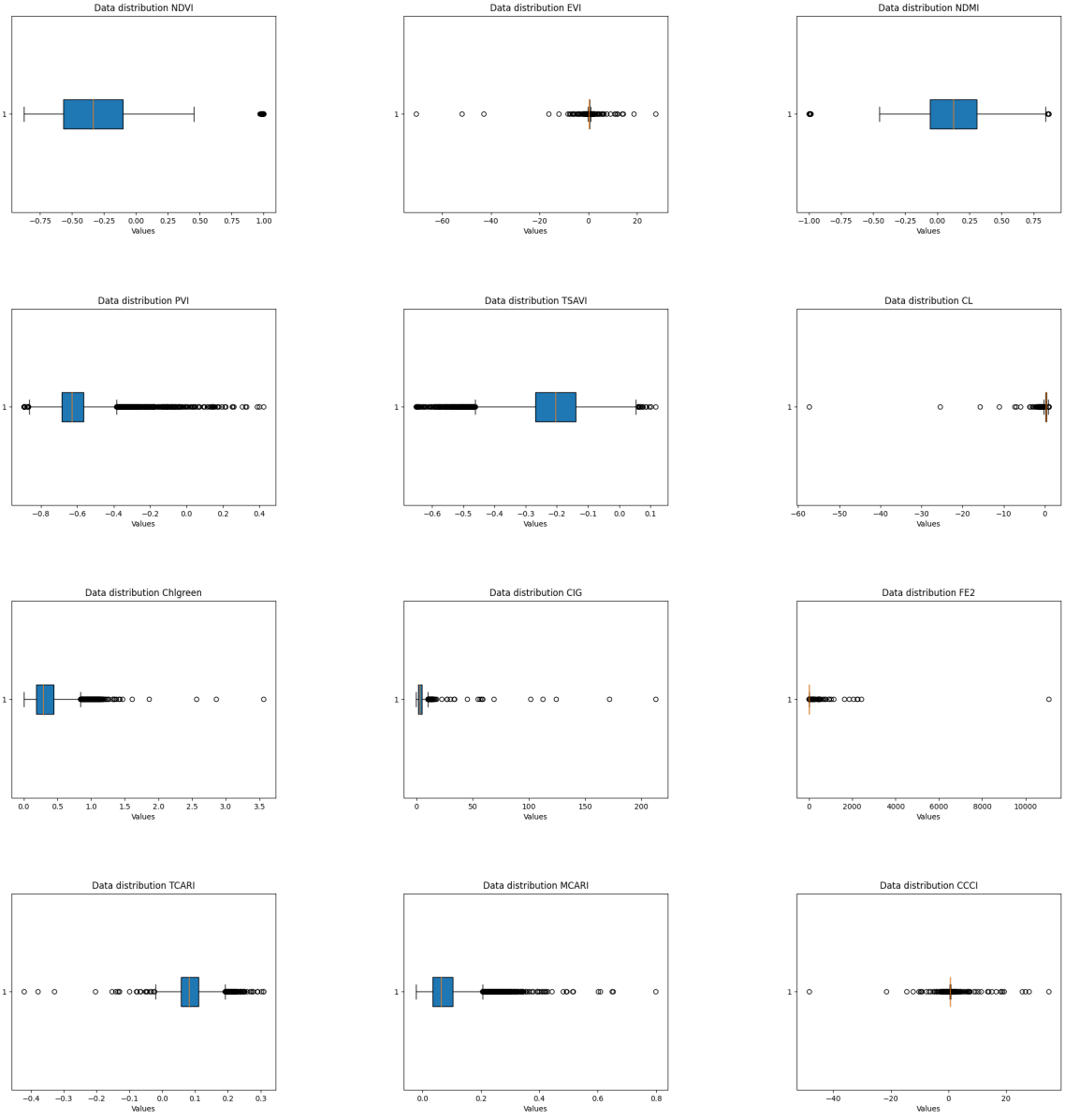


Figure A.21: Twelve of the nineteen SVI box-plots from `dataset` that have less than 100 outliers.

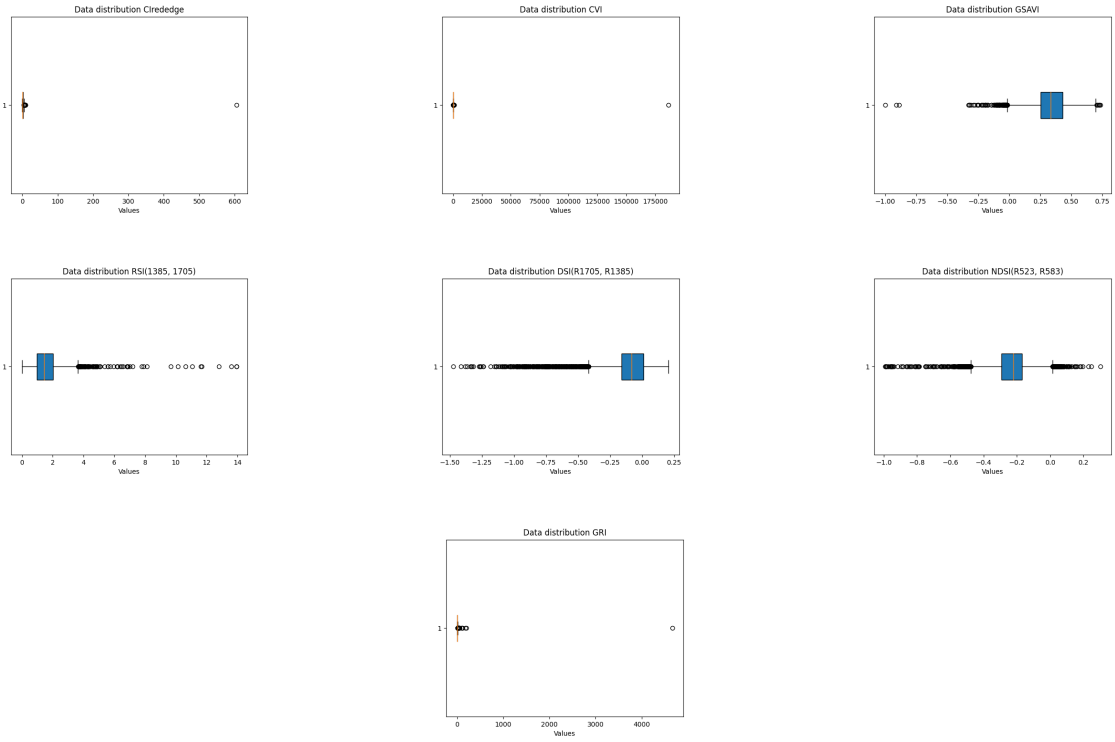


Figure A.22: Seven remaining SVI of the nineteen present in `dataset` that have less than 100 outliers.

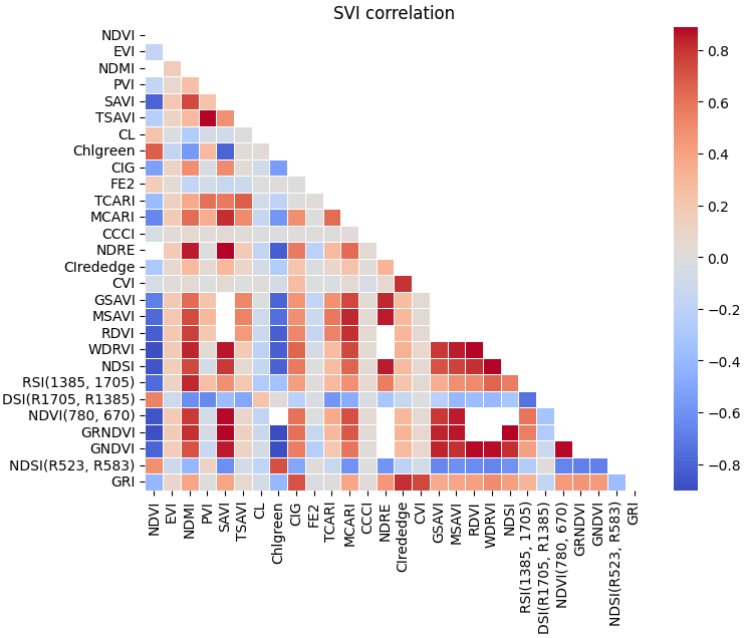


Figure A.23: Correlation matrix between SVI present on `dataset`. Note: squares omitted present a correlation higher than 0.9.