From LiDAR data to vegetation biophysical variables

Author: Pau Ventura Rodríguez

A thesis submitted for the degree of

Matemátiques Computacionals i Analítica de Dades

Advisor: Jordi Gonzalez Sabaté

2024

Acknowledgements

It has been an absolutely pleasure to work on this project with my two friends and classmates Èric Sánchez and Alejandro Donaire, probably the biggest achievement of the Functionarios' team. As part of the team, I would also like to thank Francesco Tedesco for his contributions on the latest stages of the project.

We have been working on this project thank to the amazing GFIRE team from the Computer Architecture and Operative Systems department in Universitat Autònoma de Barcelona, with a special thanks to the cheery and friendly heads, Dr. Anna Cortés and Dr. Tomàs Margalef for letting us join the team, to Antonio Espinosa for giving the best technical advice, to Paula Sánchez and Irene Gonzalez for their help and company during the hot summer days and to Dr. Carles Carrillo for his unconditional assistance, answering bore mails at all stages of the project.

Finally, I extend my sincere gratitude to the Institut Cartogràfic i Geològic de Catalunya (ICGC) for their collaboration in providing essential data for our research project.

Abstract

This project introduces a comprehensive framework using LiDAR data and machine learning techniques for predicting a range of biophysical variables. The framework is specifically tested on Canopy Cover. This research involves extensive data processing, including preprocessing and feature selection, to efficiently utilize LiDAR and additional geographical data. The study demonstrates the efficacy of the model in predicting Canopy Cover with high accuracy, reaching a R^2 score of 0.88 and a MAE of 6.47 on the whole terrain of Catalonia, highlighting its potential application in forest management and environmental monitoring. Further work suggests improvements in model accuracy and the exploration of multi-modal neural network architectures for enhanced prediction capabilities.

Contents

Chapter 1

Introduction

1.1 Problem Contextualization

Biophysical variables are fundamental indicators that quantify various aspects of the natural environment, ranging from atmospheric conditions to vegetation characteristics. They encompass a wide array of parameters that characterize the physical and biological properties of ecosystems. These variables include but are not limited to vegetation indices, soil moisture content, land surface temperature, and canopy structure [55]. They serve as key metrics for monitoring ecosystem health, biodiversity, and climate change impacts.

The accurate prediction of biophysical variables holds significant implications for various societal and environmental concerns. Understanding vegetation dynamics, for instance, is crucial for assessing carbon sequestration rates, identifying areas susceptible to wildfires, and managing natural resources sustainably.

In this study, we focus specifically on the prediction of Canopy Cover (CC). The Canopy Cover is defined by the ICGC (Institut Cartogràfic i Geològic de Catalunya) as a percentage corresponding to the sum of the areas that the canopies occupy, for a certain region. The canopies are only those of the living and normal-sized trees ($\geq$ 7.5 cm in diameter) [24]. This is the definition that we use throughout this study and, in essence, it is a measure of tree density.

Knowing this value is readily applicable to a broad range of problems, such as forest management, air pollution mitigation, carbon storage or wildfire simulations [13], the latter being our main motivation.

There are many ways of estimating those biophysical variables, which I will get into in the next section, but every of them has its own flaws. One of the most common ones is the processing time that the estimation takes, making them not a good fit for cases where an immediate prediction is needed such as wildfire simulations. Another big flaw is that there is no unified way of calculating all biophysical variables. Each one has its own different procedures. My study intends to provide a solution for those two flaws, developing a methodology for processing LiDAR data in a generalized manner to extract features that can be used to train machine learning models for predicting various biophysical variables and yielding in-real-time predictions.

I have been working on this subject since January of 2023, being a collaborator in the GFIRE project of the CAOS department in Universitat Autònoma de Barcelona. The main goal of this project is to predict wildfire spread evolution in time using a software called FARSITE. This tool can be used by firefighters to rapidly control and mitigate the fire, reducing the risks and minimizing the damages. However, FARSITE requires many inputs, including topography, weather, Fire Behavior Fuel Models, fuel moisture, and canopy characteristics [40].

Some institutions have already generated maps estimating this canopy characteristics, which get updated each 5-6 years due to the high amount of resources it takes. This results in low reliable information, as the terrain is constantly changing (consider for instance seasonality changes). To ensure the best FARSITE predictions, the variables must be up to date with the current environment. For accomplishing that, the firefighters are given an helicopter with a LiDAR system to take measurements of the area which, after being processed, can lead to in-real-time estimations of that region's variables which will then be fed into the FARSITE program. My work then is to develop an optimized software that is able to process that data in a fast way in order to extract useful information and use it for accurately predicting the given variable (in this case Canopy Cover,

but works as a framework and thus can be extended to multiple other variables). After achieving a good performance and deciding to write a scientific article on the subject, I chose to use also the topic for this thesis.

1.2 Objectives

The main goal of this project is to develop a methodology for processing LiDAR data in a generalized manner to extract features that can be used to train machine learning models for predicting various biophysical variables, providing a rapid but accurate prediction for cases where urgent estimations are required such as predicting fire spreads. Furthermore, I also aim to develop a software that implements a pipeline which is able to yield the predictions given LiDAR data as input, with parallelization capabilities and output format flexibility.

FARSITE is used to predict where will the wildfire spread the most and choosing where to invest the firefighter's human and material available resources. The wildfire spreads very fast, on average at 23km/hour [56] and thus calculating the next fire state becomes a time challenge. FARSITE itself has its own costly processing time, and thus for ensuring a useful prediction of the spreads the software must have the required inputs as soon as possible. This establishes two fundamental goals to this project: the model should be as accurate as possible, while having a significantly low processing time.

There also are specific objectives aimed at facilitating the development of a robust methodology and software solution:

Firstly, the research endeavors to establish the foundational capability to ingest and interpret LiDAR data effectively. This involves implementing algorithms and techniques to read and parse LiDAR data formats, enabling seamless integration into subsequent processing stages.

Secondly, the research aims to advance the processing capabilities of LiDAR data by devising methodologies to extract pertinent features. These features are essential for capturing meaningful information about the vegetation structure and landscape characteristics, which are crucial inputs for subsequent modeling

efforts.

The primary focus of the research lies in constructing and training a predictive model capable of leveraging LiDAR-derived features to rapidly and accurately predict canopy cover. The model development process entails the exploration of various machine learning algorithms and techniques, with an emphasis on optimizing predictive performance while ensuring computational efficiency.

Furthermore, the research endeavors to conduct comprehensive evaluations and comparisons between the predictions generated by the developed model and the estimations provided by authoritative sources such as the Institut Cartogràfic i Geològic de Catalunya (ICGC). This comparative analysis serves to validate the efficacy and reliability of the predictive model in real-world scenarios.

Lastly, the research aims to provide insights into the computational requirements and time estimations associated with the prediction process. By analyzing the computational complexity and performance characteristics of the predictive model, the research seeks to offer valuable guidance on resource allocation and optimization strategies for efficient prediction workflows.

1.3 Work structure

This section details how each section will be approached.

The project commences with an Introduction section, which explores the contextualization of the problem, defines the objectives, and outlines the overall structure of the work. It emphasizes the importance of the research topic and explains the rationale behind its selection.

Following the Introduction, the Literature Review section examines previous research on similar topics, highlighting the differences and innovations proposed by this project.

Next, the Experimental Data section describes the types of data available, data collection techniques, feature engineering processes, and data preprocessing

requirements essential for model development.

The Methodology section follows, detailing the models used, optimization methods, evaluation metrics, and feature selection process. Additionally, the workflow pipeline is outlined in this phase, as well as the Execution phase, where practical aspects of model implementation are discussed, including cluster usage, memory management, parallelization strategies, and considerations related to execution time.

Results are then presented, outlining the findings from the models, evaluations of simple models, outcomes of feature selection, and comprehensive training results, along with interpretations.

The subsequent Discussion section explores insights on the implications and limitations of the results, encouraging a discussion on the progress made and recognizing the model's constraints.

Finally, conclusions are drawn, summarizing the key research findings, emphasizing their significance and potential implications, while also setting the stage for future work and enhancements.

Chapter 2

Literature Review

Performing a literature review before starting research is crucial for several reasons: it helps identify existing knowledge and gaps in the field, preventing duplication of effort; it provides context and background, aiding in the formulation of research questions and hypotheses; it informs methodological approaches by highlighting best practices and potential pitfalls; and it establishes the research's significance, demonstrating how it contributes to the broader academic conversation.

In this chapter I will first introduce the most repeated and common approach on how to estimate Canopy Cover from LiDAR data along with its limitations. Then I will review a less known and promising variation of the first approach that seems to improve the results. Following that, I will write about the state of the art on adding image data to supplement the LiDAR input data and thus enchancing the predictions. Additionally, I will introduce the newest approaches that present solutions based on using machine learning and artificial intelligence. After that, I will present how was the groundtruth that we will be working with calculated, followed by some software that implements interesting algorithms and ending up with the conclusion of this literature review.

2.1 Height based approach

The methodology for estimating canopy cover from LiDAR point clouds generally follows a well-established sequence of steps.

Initially, a Digital Elevation Model (DEM) is created. A DEM represents the bare ground topographic surface of the Earth, excluding trees, buildings, and other surface objects [53]. This foundational step is crucial as it sets the stage for further analysis by providing a reference surface.

Following the creation of the DEM, the next step involves calculating the normalized heights of each point in the dataset, which means finding an estimation of the real heights of the points above the ground. This process is straightforward once the data has been georeferenced. The normalization is essential for distinguishing between ground and non-ground points based on their height above the DEM.

At this point, it becomes necessary to focus on the first returns from the LiDAR sensor. These first returns are critical because they typically represent the canopy, while subsequent returns penetrate deeper into the vegetation and do not provide additional useful information for canopy estimation. This filtering helps in isolating the relevant data for canopy cover analysis.

The subsequent step involves discriminating points that belong to the canopy. This is typically done by setting a height threshold above which points are classified as canopy. However, this approach presents several challenges.

One significant issue is the need to manually tune the height threshold, which can vary widely depending on tree species, age, and geographical zone. For instance, different studies have used many threshold values such as 2 meters[30], 7 meters[45] , 1.5 meters [38] , and 66% of local tree height[14]. Some researchers opt for a multi-layer approach, defining different canopy layers with upper and lower thresholds, such as ranges from 0.6096 meters to 3.048 meters, 3.048 meters to 6.096 meters, and so on [36]. This variability highlights the complexity of

accurately defining canopy thresholds across different environments.

Moreover, this method is primarily effective in forested and non-urban areas where the landscape consists mainly of trees. In urban areas, buildings and other structures can be mistakenly classified as canopy. To address this, some studies combine techniques like Principal Component Analysis (PCA) to differentiate between planar surfaces (buildings) and non-planar surfaces (trees). Additionally, multispectral satellite imagery can be used to calculate the Normalized Difference Vegetation Index (NDVI) for each LiDAR point. NDVI helps distinguish vegetation from non-vegetation, allowing for the reclassification of points initially misidentified as buildings if they exhibit high NDVI values [34].

Another limitation of this method is the assumption that all trees within the study area are of the same species and age. Variations in tree morphology due to species diversity and age differences can lead to discrepancies in canopy height, complicating the classification process.

Once the canopy points have been accurately discriminated, the final step is to calculate canopy cover. This is done by determining the proportion of filtered points within a given tile that belong to the canopy. This step provides a quantitative measure of canopy cover, which is essential for various ecological and environmental assessments.

2.2 Vegetation based approach

Instead of focusing on the height of the points, some other studies focus on the classification of the points, specifically leveraging ICESat GLAS data. The authors focus on understanding spatial patterns of canopy cover across different forest types globally. They analyze the canopy cover distribution by biome and forest type, revealing distinct patterns such as the bimodal distribution in evergreen broadleaf forests and the trimodal distribution in deciduous broadleaf forests. The footprint-level canopy cover fraction is calculated as the ratio of vegetation return to total waveform energy, adjusted by a global average reflectance ratio of 1.5. This process includes recursive analysis to refine initial estimates, accounting for slope effects and rejecting outliers. Methodologically, the authors

utilize ICESat GLAS-derived canopy cover estimates, which they compare with MODIS VCF estimates to validate their findings. This approach highlights the variations in canopy cover across different biomes and emphasizes the importance of ICESat GLAS data in capturing detailed forest canopy structures globally. [51]

2.3 Supplementing the prediction using imagery data

Some studies, in order to support LiDAR data and get more accurate predictions, choose to add imagery data.

To improve classification quality, a study conducted in a suburb of Melbourne, Australia, proposed a method that fuses LiDAR point cloud data with multispectral satellite imagery. This approach begins by associating each LiDAR point with spectral information from co-registered satellite imagery, calculating the normalized difference vegetation index (NDVI) for each point to correct tree points misclassified as buildings. Region growing of tree points, incorporating NDVI values, refines the classification, and the identified tree points are then used to generate a canopy cover map. Experimental evaluation using airborne LiDAR and WorldView 2 imagery demonstrated that integrating multispectral imagery significantly enhances the accuracy of tree canopy cover mapping in urban environments. [34]

Another study aimed to develop a canopy cover estimation model using Landsat 8 OLI imagery alongside LiDAR data. Landsat 8 OLI imagery was preprocessed with geometric and topographic corrections to eliminate distortions and effects of illumination. The study utilized vegetation indices derived from Landsat 8 OLI, including NDVI (Normalized Difference Vegetation Index), GNDVI (Green Normalized Difference Vegetation Index), and SRVI (Simple Ratio Vegetation Index), which provided essential information on vegetation health and density. These indices were critical in integrating with LiDAR-derived canopy cover estimations. The combined data facilitated the creation of a regression model

that related Landsat-derived vegetation indices to LiDAR's First Return Canopy Index (FRCI). This integration enhanced the accuracy of large-scale canopy cover estimation by leveraging the extensive spatial coverage of Landsat imagery and the detailed three-dimensional information from LiDAR. The model's effectiveness was demonstrated with an equation $FRCI = 2.22 + 5.63Ln(NDVI)$, achieving an R^2 of 0.663, highlighting the significant role of multispectral imagery in supplementing LiDAR data for canopy cover mapping. [45]

Coupland et al. utilized both historical aerial photographs and recent LiDAR data to monitor tree canopy cover (TCC) changes at the University of British Columbia's Vancouver campus. Historical aerial photos from 1949 were obtained from UBC's Geographic Information Centre and manually scanned at high resolution. These images were georeferenced and rectified in ArcMap using ground control points and then stitched together to cover the study area. For recent data, LiDAR was collected in 2015 with high point density, allowing for detailed canopy height models after removing buildings. TCC was assessed by comparing these datasets within a grid of 0.05 ha analysis polygons, and the methods were validated using modern aerial photos from 2015. This combination of historical imagery and high-resolution LiDAR allowed the researchers to effectively track changes in urban canopy cover over a period exceeding 50 years, demonstrating the method's suitability for long-term environmental monitoring despite the disparity in data types. [6]

Researchers have also used Digital Hemispherical Photography (DHP) was to validate canopy cover estimates derived from LiDAR data. DHP involves taking photographs from a hemispherical camera oriented upwards, capturing the projection of tree parts like leaves, branches, and trunks. These images are processed to estimate canopy cover by analyzing the proportion of sky obscured by vegetation. This ground-based method is widely used for calibrating and validating remote sensing data, as it provides a detailed and direct measurement of canopy cover from beneath the canopy. In this study, the researchers compared the canopy cover estimates obtained from DHP with those derived from LiDAR point clouds, which included metrics accounting for the distribution and over-

lap of vegetation points within the data. The comparison showed a significant correlation, indicating that DHP is an effective tool for validating LiDAR-based canopy cover models in tropical forest plantations. [38]

2.4 Machine Learning Models

Instead of using the height to identify the canopy, many studies use it to generate aggregate features which will be used to train a model that will have the canopy cover value as output. Models of different complexity are used.

The most simple machine learning model employed is linear regression by Narine et al. in 2019. The features used as inputs for the linear regression models were derived from a set of ICESat-2 photon-counting lidar (PCL) metrics that represent canopy structure and density, which were generated using existing airborne lidar data from the Sam Houston National Forest in Texas. The features included canopy height metrics, such as the 90th percentile height (P90); canopy cover metrics, calculated as the fraction of PCL returns above a certain height threshold; vertical distribution metrics, like the height of median energy (HOME); and gap fraction, representing the proportion of gaps within the canopy. These features were extracted from 100-meter segments along simulated ICESat-2 tracks by replicating PCL data from existing airborne lidar data in the Sam Houston National Forest, Texas, and adding photon noise to mimic real ICESat-2 conditions. The derived features were then used to train and validate the linear regression models, which demonstrated strong predictive performance with R^2 values up to 0.93 for canopy cover in noise-free scenarios, highlighting the effectiveness of these lidar-derived metrics in capturing forest canopy structure and density. [29]

Other studies add complexity to the predictiors of the linear regression using height percentiles and non-cumulative bin coefficients as input features, extracted through a detailed process. Initially, raw LiDAR ground and vegetation points were normalized to ensure ground points had zero elevation, while vegetation points represented relative heights above the ground, excluding those below 1.5

meters to match digital hemispherical photograph (DHP) measurements. The remaining vegetation points were divided into ten non-overlapping height bins at 10% intervals of the maximum vegetation height, quantifying the relative area covered by each layer. For each DHP setup point, a ten-meter circular radius buffer was created to clip the layered vegetation points, and height percentiles were generated. These percentiles and bin coefficients were derived using a "what-if" analysis tool, allowing for the construction and testing of the regression model by comparing DHP and LiDAR-based canopy cover estimates. This method demonstrated that LiDAR-derived metrics, when applied in a regression model, could reliably estimate canopy cover, emphasizing the importance of parameter selection and model calibration for different forest classifications. [38]

Not only linear regresors are used for this kind of task but also more complex models like Random Forest (RF) models were developed using both custom-processed ICESat-2 data and ATL08 parameters to estimate canopy cover. The input features for these models included a variety of canopy cover metrics such as the percentage of photons above specific height thresholds (e.g., 2 m and 4.6 m), maximum height, and various percentile height metrics. These features were extracted from the LiDAR data by processing the photon count and categorizing them into different canopy and ground return classes. The study demonstrated that RF models slightly outperformed traditional linear regression models in some instances, achieving lower RMSE values. For instance, the RF model for predicting canopy cover above 4.6 m in the Southern US forests exhibited a lower RMSE compared to linear regression, highlighting its effectiveness in capturing non-linear relationships within the data. [28]

2.5 ICGC's method

Our groundtruth was provided by ICGC and thus it is key to understand the process of how did they estimate the Canopy Cover.

In this case, the process of calculating canopy cover integrates field measurements, LiDAR data, and biophysical modeling. Initially, values are obtained by

applying allometric equations to each measured tree within the sampling plot, relating crown diameter to the diameter at breast height (DBH) specific to each species, with the calculated canopy cover potentially exceeding 100% due to crown overlap. To generate 20-meter resolution maps of biophysical tree variables, LiDAR data is processed to classify vegetation and terrain points accurately and to calculate structural metrics of forested areas. These allometric equations are applied to individual trees using forest inventory data to estimate biophysical variables. Land cover maps are processed to assign corresponding tree cover to each 20x20 meter cell. Statistical models that best fit the forest inventory are then calculated based on LiDAR metrics, and the cartography is generated by applying these models to the LiDAR metrics according to the species and tree categories indicated in the land cover map. Finally, confidence intervals are calculated to ensure the accuracy of the estimates. This comprehensive approach allows for detailed and accurate mapping of canopy cover, accounting for species-specific growth patterns and the structural complexity of forested landscapes. [24]

This methodology, while detailed and comprehensive, has several downsides that can contribute to extended timeframes for estimating canopy cover. The process involves multiple complex steps, including the initial classification and processing of LiDAR data to accurately distinguish between vegetation and terrain points, which is computationally intensive. The application of species-specific allometric equations to individual trees using forest inventory data is also labor-intensive, requiring precise data collection and extensive fieldwork. The integration of various data sources, such as forest inventories and land cover maps, necessitates careful alignment and calibration to ensure accuracy, adding to the time required. Additionally, developing and validating statistical models to fit the forest inventory data involves significant analytical effort, and generating cartographic outputs from these models is both time-consuming and resource-intensive. Furthermore, calculating confidence intervals to ensure the accuracy of the estimates adds another layer of complexity and time. Overall, the meticulous nature of each step, combined with the necessity for high precision and the integration of diverse data sets, contributes to the lengthy duration required to produce reliable canopy cover estimations.

2.6 Built in software

There exist some already built software that help on the task of processing LiDAR data to obtain different metrics, such as he Forest Tools R package [37], which offers functions to analyze remotely sensed forest data. It is useful for detecting tree tops or building a canopy height model. Can also detect tree crowns, which can be used to calculate the canopy cover. However, the algorithm used for finding the tree crowns is still defining a threshold for the height at which the canopy begins.

Another well known software is Tiffs: Toolbox for Lidar Data Filtering and Forest Studies [4], which is used to process lidar data, generate digital elevation models, digital surface models, and canopy height models, and extract individual tree structural information, including tree height, crown area, and biomass. It includes functions for data tiling, point cloud filtering, and simulating waveforms for validation purposes, offering a user-friendly interface with efficient visualization capabilities.

2.7 Literature conclusion

Although all the studies mentioned above reach decent performance with a variety of different models (more on that in the Discussion section), each one has its own flaws: either some parameter has to be tuned such as the height from which we start considering a point to belong to the canopy, or the method only works on forest and non-urban areas as the model does not distinguish between trees and buildings, or rather the model requires extra information such as satellite imagery, or even the model is excessively complex and takes a long time to extract precise estimations. Moreover, all of them meet the same issue: they are only built for detecting Canopy Cover. In this project we propose an automated framework based on Machine Learning and Artificial intelligence that not only depends on LiDAR and static data, but also does not need parameter tuning and is able to adapt and predict many biophysical variables regardless of the

environment, reaching an outstanding performance while yielding fast predictions. Furthermore, we will compare the performance of our model with the performance of some of the models previously described, such as the vegetation based models or the height based models.

Chapter 3

Experimental Data

In this chapter I will be focusing on the data that was used throughout this thesis.

First, I will lay the theoretical groundwork essential for understanding the methodologies and concepts underpinning the data that I will be working with, mainly explaining what is the LiDAR data, how is it collected, the attributes that it contains, how is it structured and stored, along with its specifications. I will dedicate an entire section for this purpose as LiDAR data was our main source of information which consolidated the bulk of the processed dataset and it is important to fully understand every aspect of it. Then I will provide a comprehensive overview of all the other data sources that were used, their purpose, source, resolution, visualization, alternatives that were considered and structure. Following that section, I will write about the software tools utilized for data manipulation in my research. Additionally, I will describe the various software applications employed to process, analyze, and manage the LiDAR and supplementary geographical data, ensuring the accuracy and efficiency of our predictive modeling efforts. Afterwards, I will enumerate all the datasets used during the project, built and extracted from the available data. Finally, it is important to remark the features that were extracted from the data, how were they calculated and what do they represent.

3.1 Theoretical framework

First, it is important to properly define what LiDAR is, as I am extracting data from that source. LiDAR, which stands for Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth. This method generates precise, three-dimensional point cloud about the region measured and its surface characteristics. A LiDAR instrument consists of a laser, a scanner, and a specialized GPS receiver. The data collected by LiDAR systems are used for various applications, including mapping both natural and manmade environments, supporting activities such as inundation and storm surge modeling, hydrodynamic modeling, shoreline mapping, emergency response, hydrographic surveying, and coastal vulnerability analysis [32].

LiDAR is widely used in many engineering fields and it is becoming increasingly available, being free in many cases. Apart from its availability, LiDAR data is a good choice to predict the Canopy Cover for other reasons. Firstly, it can be easily treated using Data Analysis tools and can be fed to Machine Learning algorithms. Second, if it comes with some preprocessing done, attributes such as intensity or classes are also available, which are extremely useful. Third, it can be collected fast. This is especially useful for real-time emergencies if the data is being used for wildfire simulations. Finally, the LiDAR technology is improving, and we expect to have higher density point clouds in the future, which will very likely improve the performance of models like the one presented here.

In order to collect LiDAR data, the first step is to plan the flight: the area to be scanned is decided, the altitude (which determines the number of LiDAR returns per pulse), and the velocity. The flight lines for the pilot to follow are also planned, ensuring that parallel lines are sufficiently close together to provide overlap between the .las files. Once the flight is completed, the trajectory is processed, and the points are georeferenced. Some corrections are made to improve georeferencing accuracy, and noise (such as birds) is removed. Once this is done, the points are classified.

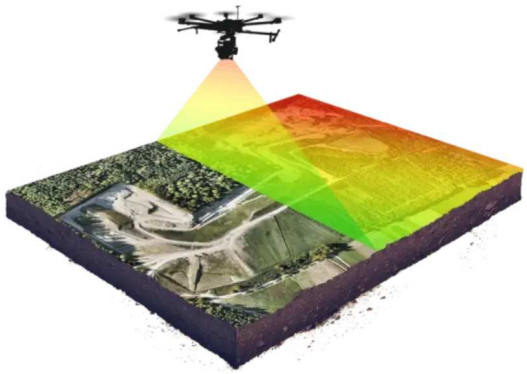


Figure 3.1: Lidar data gathering representation.

Once the LiDAR data is collected, it contains the following attributes (in our context): [23]

- **X**: Denotes the spatial coordinate in the east-west direction.
- **Y**: Denotes the spatial coordinate in the north-south direction.
- **Z**: Contains the height of the point. However, it does not correspond to the distance between the ground and the measurement but between the geoid EGM08D595 and the measurement in the gravity direction (orthometric heights).
- **Intensity**: The integer representation of the pulse return magnitude. This value is optional and system specific. Takes values from 0 to 255. When compared with the Canopy Cover, Figure 3.2 shows that Intensity is somehow correlated to Canopy Cover.

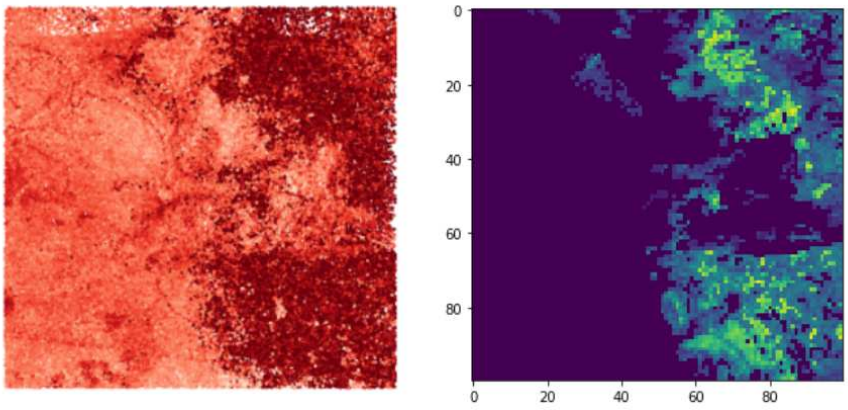


Figure 3.2: Intensity (left) and Canopy Cover (right) colormaps of the same area.

- **Return Number**: The Return Number is the pulse return number for a given output pulse. A given output laser pulse can have many returns, and they must be marked in sequence of return. The first return will have a Return Number of one, the second a Return Number of two, and so on up to five returns. When visualizing the return number, Figure 3.3 shows that this feature provides important information to the terrain irregularities, which might mark the spot of trees.

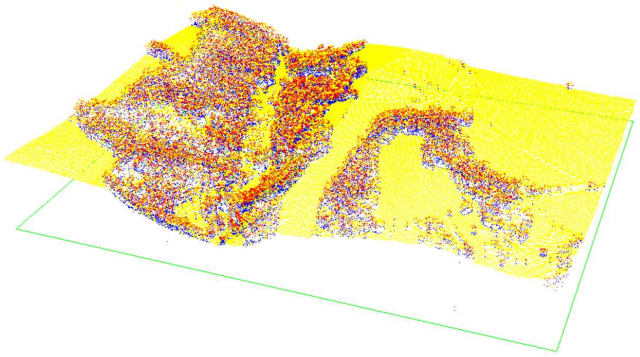


Figure 3.3: Return number plotted: red = 3, blue = 2, yellow = 1.

- **Number of Returns**: The Number of Returns is the total number of returns for a given pulse. For example, a laser data point may be return

two (Return Number) within a total number of five returns. Figure 3.4 shows a representation of the different pulses registered by LiDAR from a same stream.

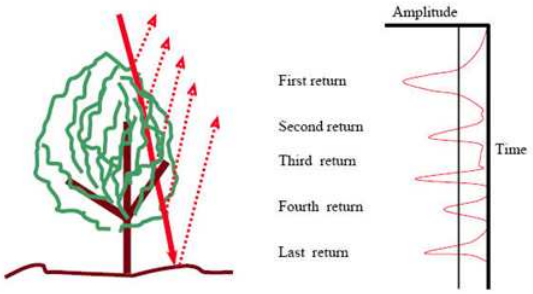


Figure 3.4: Representation of the different pulses registered by LiDAR from a same stream.

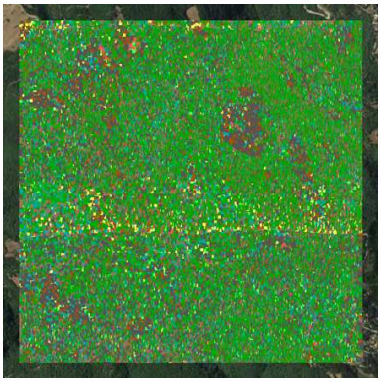
- **Scan Direction Flag**: The Scan Direction Flag denotes the direction at which the scanner mirror was traveling at the time of the output pulse. A bit value of 1 is a positive scan direction, and a bit value of 0 is a negative scan direction.
- **Edge of Flight Line**: The Edge of Flight Line data bit has a value of 1 only when the point is at the end of a scan. It is the last point on a given scan line before it changes direction.
- **Classification**: Defines the type of the point. The following table describes the different classifications assigned to the LiDAR data that we worked with, although it can change depending on the data's source.

Classification	Description
1	Default
2	Ground
3	Low Vegetation
4	Medium Vegetation
5	High Vegetation
6	Building
7	Low Point
8	Model Keypoints
11	Air points
13	Other ground
14	Wires
15	Transmission Tower
18	Other towers
135	Noise

- **Scan Angle Rank**: The Scan Angle Rank is a number with a valid range from -90 to +90. The Scan Angle Rank is the angle at which the laser point was output from the laser system including the roll of the aircraft.
- **gps time**: The GPS Time is the time tag value at which the point was acquired.

Figure 3.5 shows how LiDAR cloud point is able to represent an area using single points, distributed in different classes. If the same area is visualized using a satellite, the similarities between bare eye and LiDAR data visualization are very clear.

(a) Google Earth visualization of a 2km × 2km area.



(b) LiDAR visualization of a 2km × 2km area.

(c) Legend of LiDAR points.

Figure 3.5: Satellite and LiDAR visualization of the same area.

If we include the z coordinate in the visualization, the landscape is even clearer as shown in Figure 3.6. Note that the points in the sky belong to class 135 with is labeled as Noise i.e. outliers and will be eliminated.

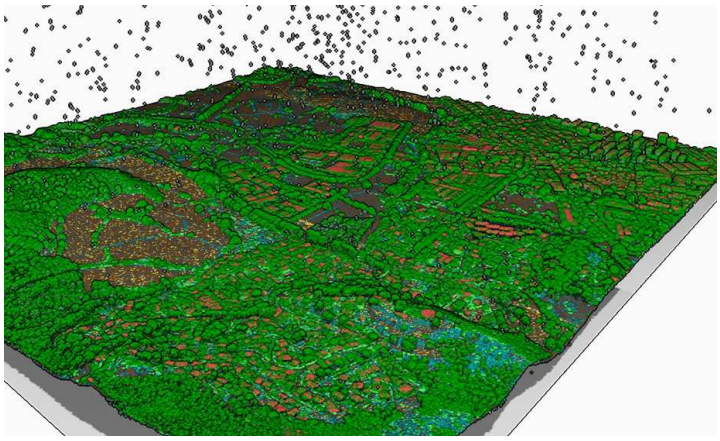


Figure 3.6: 3D LiDAR data visualization.

This LiDAR data will be used to predict Canopy Cover.

3.2 Data Gathering

For this project we required to gather as much information as possible with a requirement: it had to be static data, which means that is not drastically changing over time. This way we could always use the same map regardless of the time in which our software was used, without the need to finding an update.

The different types of data that we gathered for this project are:

1. **Canopy Cover**: Defined as a percentage corresponding to the sum of the areas that the canopies occupy, for a certain region. These are the values we want the model to be able to predict also refered to as groundtruth.
    - Source: Institut Cartogràfic i Geològic de Catalunya (ICGC). [24]
    - Image: Figure 3.7.

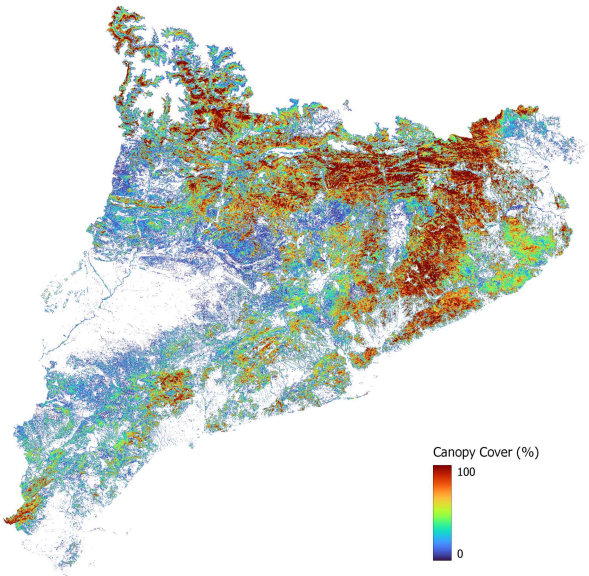


Figure 3.7: Catalonia's Canopy Cover map.

    - Resolution: 20 × 20 m. This implies that we have to adapt the information available to this resolution.

- Refference system: ETRS89 UTM 31 North, in the order Easting(X),Northing(Y), with code EPSG:25831.
- Date collected: between 2016 and 2017.
- Range of values: from 0 to 199.635. This is well over 100%, with the reason being that the definition accounts for canopy overlapping, although high values are rare. However, we limit this value to 100 as FARSITE input accepts values from 0 to 100, thus all values over 100 are set to 100.

2. **LiDAR**: It is used to calculate aggregate metrics for each 20 × 20 m section of the surface. These metrics encapsulate essential information about that area. The point cloud had been captured with LiDAR sensor, calibrated and adjusted with topographic control areas, obtaining an altimetric accuracy with a mean square error of about 6 cm in flat areas with little vegetation. Subsequently, the cloud had been classified automatically. The downloaded files are in format LAS 1.2 compressed. It can be decompressed using tools such as LASzip.

   - Source: Institut Cartogràfic i Geològic de Catalunya [23].
   - Image: Figure 3.8.

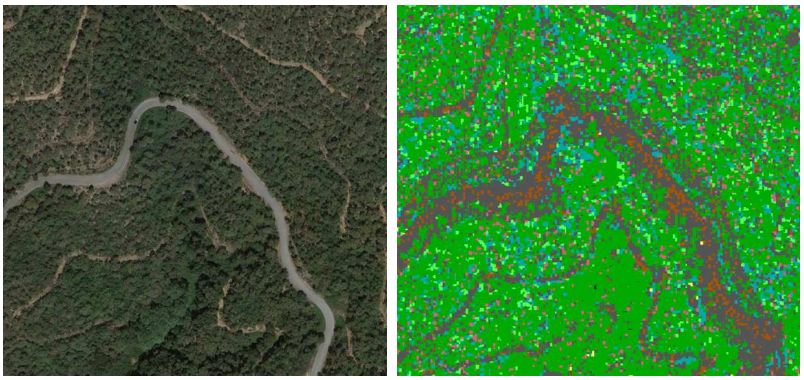
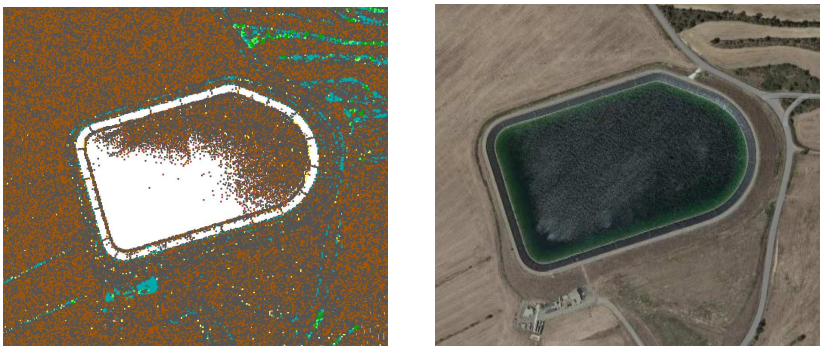


Figure 3.8: Google Earth and LiDAR comparison of the same area.

   - Resolution: It has been ensured that 95% of the blocks have a minimum density of 0.5 points/m2. The remaining 5% are blocks that

cover areas of water or that are located on the border with Aragon and France or on the coast. This means that in a typical 20 × 20 m region there are between 100 to 600 LiDAR points. However, there are some areas that contain very low density such as the observed in the figure 3.9.

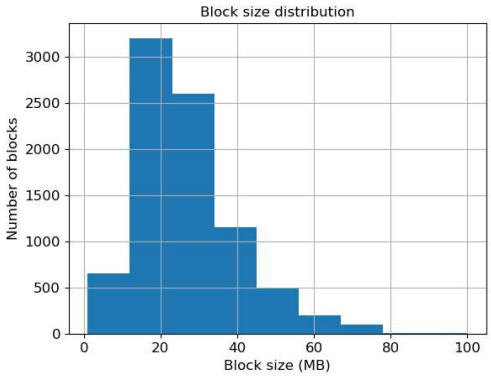


(a) LiDAR point cloud. (b) Google Maps visualization.

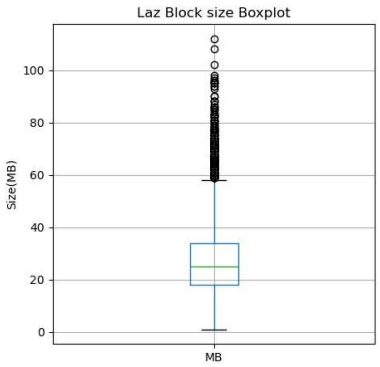
Figure 3.9: Different views of a low density area.

- Refference system: Coordinates in UTM projection zone 31 and geodetic reference system ETRS89. Orthometric altitudes and referenced to the EGM08D595 geoid.
- Attributes: the LiDAR data from the ICGC comes with some preprocessing done, which means every LiDAR point is not only associated with the x, y and z coordinates, but also with other features. The features we use are: Classification, Intensity, Scan angle, Number of returns, Return number and GPS time.
- Date collected: between 2016 and 2017.
- Structure: The LiDAR data from the ICGC is organized in discrete .laz (a compressed version of .las) chunks that measure 2 × 2 km. These often weight, without decompressing, approximately 50 MB. In this project I will often refer to these chunks as "blocks". The Spanish region of Catalonia consists of 8434 blocks, adding up to a total size of around 3TB. However, not all blocks have the same size, as denser regions such as forest will result in higher file size than for example

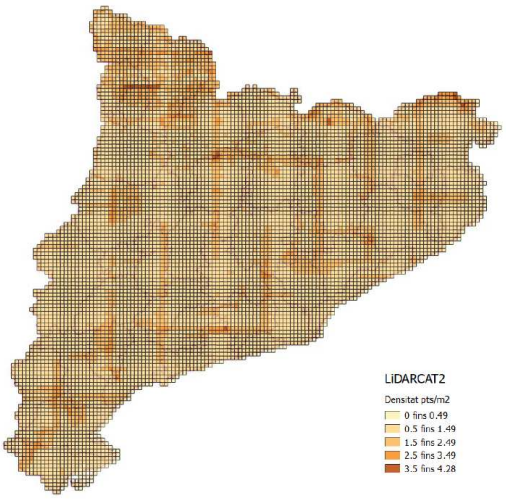
a flat area. The figure 2 shows an histogram and a boxplot of the distribution of blocks, along with a distribution of the different sizes along Catalonia.



(a) Block Size Histogram.



(b) Block Size Boxplot.



(c) Block Size Map Distribution.

Figure 3.10: Histogram, Boxplot and distribution of LAZ filesize.

3. **LandUse**: Displays a global map of land use/land cover (LULC) derived from ESA Sentinel-2 imagery. Each year is generated with Impact Observatory's deep learning AI land classification model, trained using billions of human-labeled image pixels from the National Geographic Society. The global maps are produced by applying this model to the Sentinel-2 Level-

2A image collection on Microsoft's Planetary Computer, processing over 400,000 Earth observations per year. LULC Provides information about the type of ground. There are 11 different categories and every region is classified as one of them. This classification includes categories such as crops, rangeland, built area or water. We strongly believed that the LandUse would give further information to the model about the zero valued canopy cover areas due to features such as "build area".

- Source: Impact Observatory, Microsoft, and Esri[10].
- Image: Figure 3.11

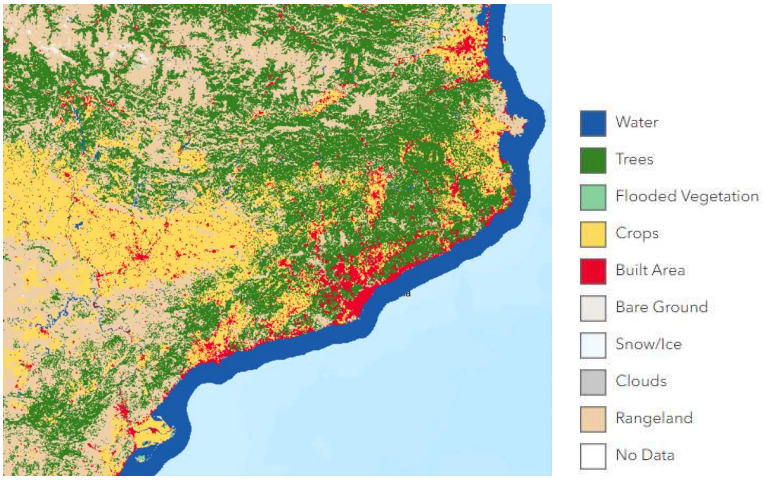


Figure 3.11: ESRI LandUse/LandCover

- Resolution: 10 × 10 m. This means that there are 4 different types of LandUse per every 20 × 20 m region. We simply choose the upper left type, given that the LandUse is fairly homogeneous. LandUse will be later one hot encoded in the following way: one column will have a number according to the amount of landuse tiles that correspond to that feature. This is, if 20×20 Canopy Cover tile contains 2 10×10 tiles of type 1 LandUse, 1 type 3 and 1 type 5, the encoding will have a 2 on the type 1 column, a 1 on the type 3 and a 1 on the type 5, the others will be set at 0. However, this encoding did not show to improve the results to we decided to keep it simple as this encoding required

further processing time. Having such a high resolution resulted in large memory usage and thus we had to implement optimizations as Catalonia had 700.000.000 tiles.

- Refference system: Web Mercator Auxiliary Sphere WGS84 (EPSG:3857)
- Date collected: 2017.

4. **Slope**: Has information about the slope of the ground.

   - Source: it is obtained using the Digital Terrain Model from the ICGC [22]. The raster is processed using geospatial processing tools, in particular, GDAL. [11]
   - Image: Figure 3.12.

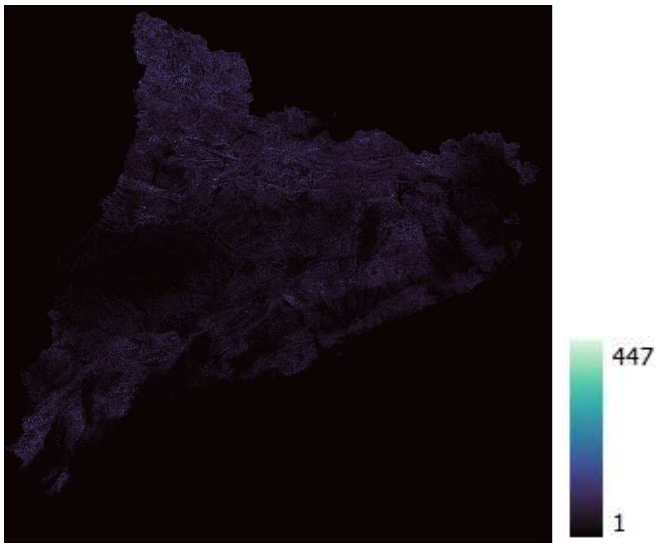


Figure 3.12: Catalonia slope's map.

   - Resolution: 20 × 20 m, so there is a direct correspondence with the Groundtruth data. It takes values from 0 to 447.
   - Refference system: ETRS89 UTM 31 North, in the order Easting(X),Northing(Y), with code EPSG:25831.
   - Date collected: 2020.

5. **Aspect**: Information about the orientation of the ground.

- Source: in a similar way to the Slope, it is obtained using the Digital Terrain Model from the ICGC [22]. The raster is processed using geospatial processing tools, in particular, GDAL. [11]
- Image: Figure 3.13.

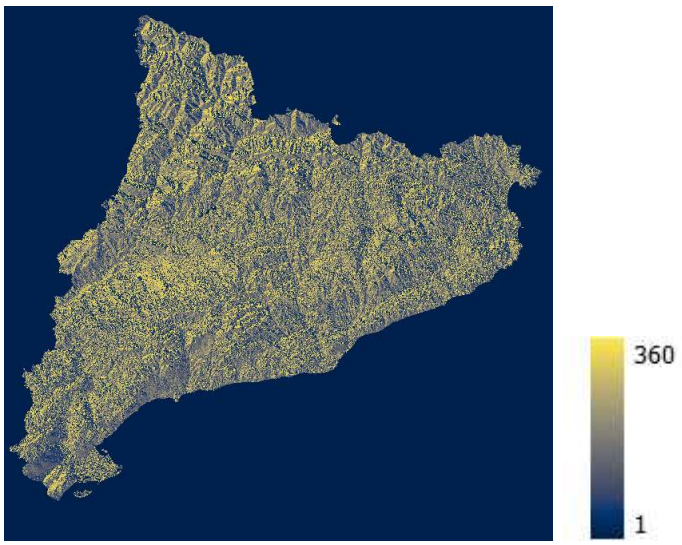


Figure 3.13: Catalonia aspect's map.

- Resolution: 20 × 20 m, so there is a direct correspondence with the Groundtruth data. It takes values from 0 to 360 with outliers at -9999.
- Refference system: ETRS89 UTM 31 North, in the order Easting(X),Northing(Y), with code EPSG:25831.
- Date collected: 2020.

At the beginning we considered using fuel maps such as Burgan's [47] or Anderson's [1], as they showed to significantly increase the accuracy of our model (see Figure 3.14) due to their indicators for zero valued canopy cover as seen in Figure 3.2, but we discarded that idea as fuel maps are not often updated and thus we would not have that information when using the framework in the future. On the other hand, ERIS LandUse is updated each year and has similar information so an updated version is always available.

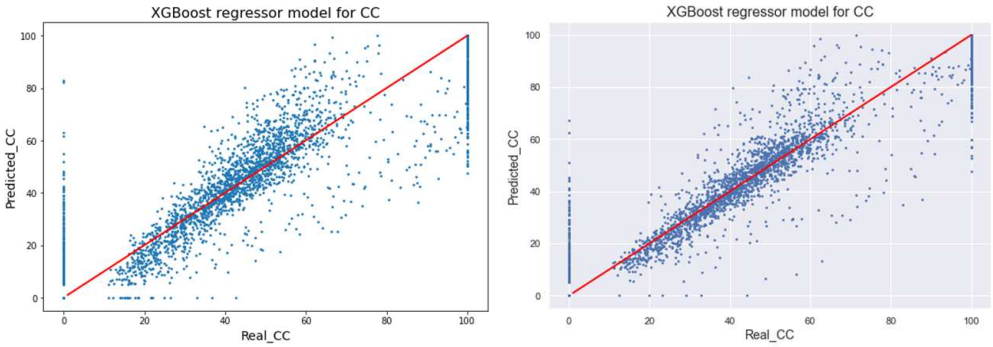
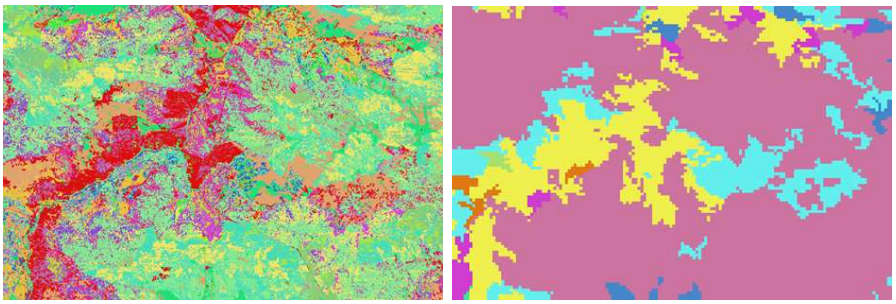
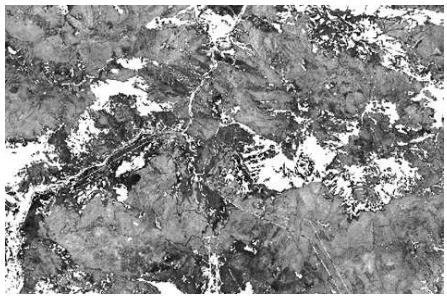


Figure 3.14: Initial model results comparison of a full block of 2×2km without including the fuel information (left) and including the fuel information (right).



(a) Burgan fuels model.

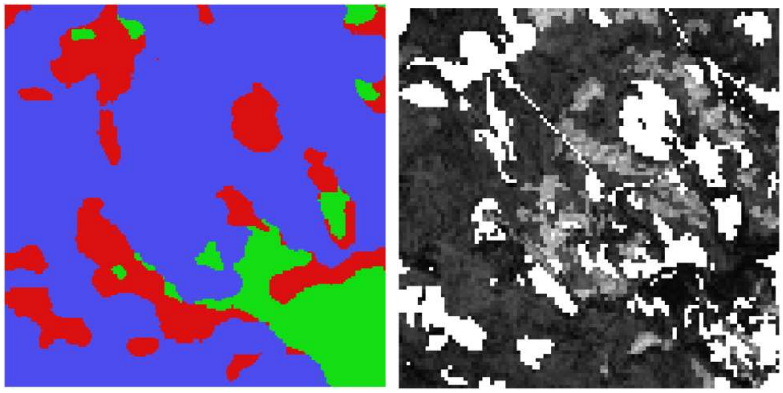
(b) Anderson fuels model.



(c) Canopy Cover.

Figure 3.15: Fuels model comparison with canopy cover.

We also considered other LandUse models such as GISAT [19], but discarded it as due to the low resolution (100mx100m) we could not extract much information from it as seen in Figure 3.2.



(a) GISAT LandUse. (b) Canopy Cover.

Figure 3.16: Fuels model comparison with canopy cover.

In terms of Canopy Cover, we also investigated many data sources, like Burgan Canopy Cover maps [39] provided by Previncat. The resolution was also 20 × 20m but it was not aligned with the ICGC's one as shown in Figure 3.17 so we could not one-to-one compare them.

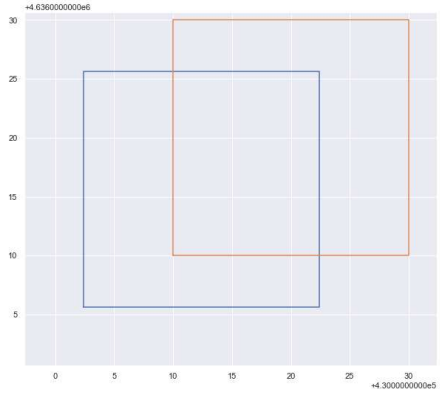


Figure 3.17: 20×20m tile situation comparison between Burgan (blue) and ICGC (orange). Note that they are not aligned.

However, we could compare the Canopy Cover distribution over a same area. Figure 3.18 shows the histogram of both Canopy Cover models over the same area.

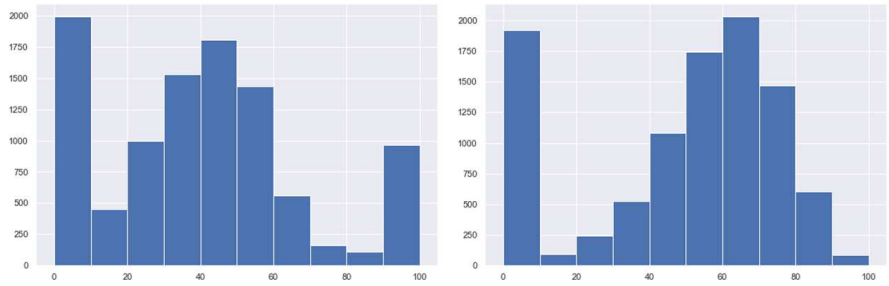


Figure 3.18: Canopy Cover (x-axis) histogram comparison between ICGC (left) and Burgan (right)

While the number of zero valued entries is very simmilar, we can see that ICGC's distribution is more centered with a lot of values close to 100, while Burgan's distribution is more right-shifted with few values close to 100.

If we compare the same area on a map in Figure 3.19, we can deeply see that phenomenon, as the ICGC's map has more area covered in cold colours with some spikes at intense red while the Burgan's model contain more hot colors.

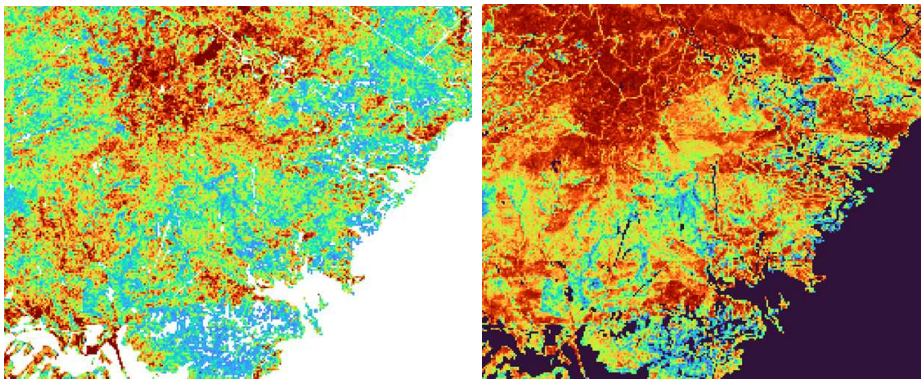
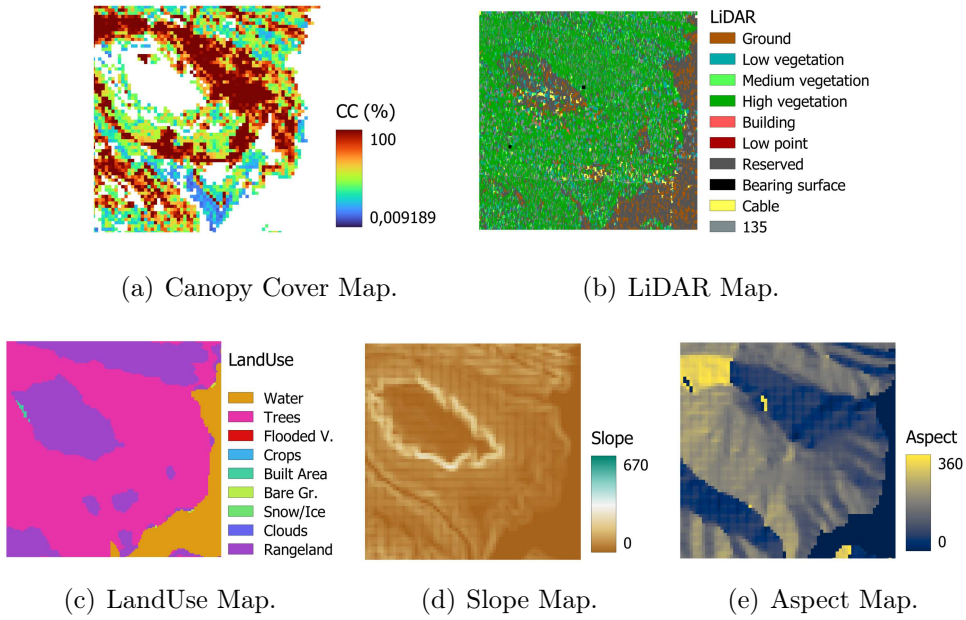


Figure 3.19: Canopy Cover map comparison between ICGC (left) and Burgan (right).

Given that the ICGC's map had more realistic values when checked it with tools such as google earth and was also aligned with other data maps such as slope and aspect, we chose to keep ICGC's one as the groundtruth.

Figure 3.20 shows all the maps that we ended up using for this project, each one of them showing a different propperty of the same area.



(a) Canopy Cover Map. (b) LiDAR Map.

(c) LandUse Map. (d) Slope Map. (e) Aspect Map.

Figure 3.20: Different input data compared with the Groundtruth for the same region.

3.3 Software used

During this project we required a deep understanding of the datasets we were working with and thus special visualization and processing tools were necessary. The used software mainly focuses on geospatial data, as it is the kind of information that we were working with.

For visualization purposes we used the QGIS software [41]. QGIS is a geographic information system (GIS) software that is free and open-source. It supports viewing, editing, printing, and analysis of geospatial data in a range of data formats.

We also used GDAL [15] in combination with QGIS for preprocessing the maps and reducing their size. GDAL is a translator library for raster and vector geospatial data formats that is released under an MIT style Open Source License by the Open Source Geospatial Foundation. It also comes with a variety of useful

command line utilities for data translation and processing.

Another important software was LAStools [42], which is a collection of 52 highly efficient, batch-scriptable, multicore command-line tools for processing point clouds. The software combines robust algorithms with efficient I/O, making LAStools the fastest and most memory-efficient solution for batch-scripted multicore LiDAR processing. It can turn billions of points into valuable products at blazing speed and with low memory requirements [43].

Google earth was also used for visualizing different areas of Catalonia from a satellite point of view, such as in Figures 3.8, 3.5. Google Earth is a computer program that renders a 3D representation of Earth based primarily on satellite imagery. The program maps the Earth by superimposing satellite images, aerial photography, and GIS data onto a 3D globe, allowing users to see cities and landscapes from various angles [58].

3.4 Datasets used

During this project we have build many datasets, each one with a different purpose and characteristics. On the beginning we build very simple datasets for visualization purposes and for understanding the data we would have to work with. A small dataset would also help us monitoring how the simple models worked and the flows they had, as well as comparing the input data with the groundtruth to gain information on which features better explained the response variable. Then, as the models got more complex we needed more data to train them so we had the urge to build bigger datasets.

All datasets are subsets of the whole Catalonia's region.

The datasets used are:

1. **Tiny dataset**.
    - Purpose: First approach to the data. Minimal representation of the

data. Understanding how the LiDAR points are distributed and doing a close up visualization to our data. Useful also for debugging. Comparing the dataset with the groundtruth and seeing which classes populate the zones with higher Canopy Cover.

- Resolution: 100 × 100 m. Represents a very small area of an entire block as seen in Figure 3.21, with only 25 tiles of 20×20 inside.

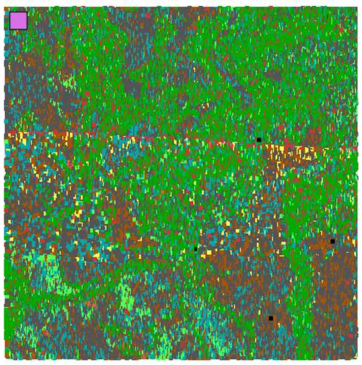


Figure 3.21: Tiny dataset size comparison with a full block of 2km × 2km.

The green areas represent the Canopy Cover as shown in Figure 3.22.

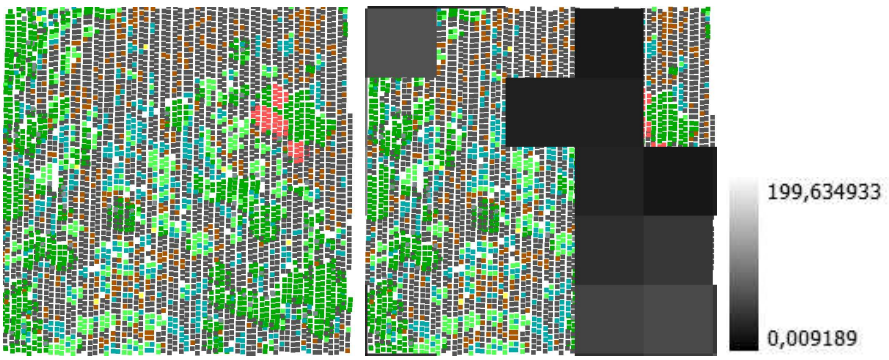


Figure 3.22: Tiny dataset visualization and Canopy Cover comparison.

2. **Toy dataset**

- Purpose: A dataset slightly larger, for testing how the simplest models perform on more data. Testing how does the algorithm adapt to irregular shaped datasets.

- Resolution: 680 × 560m. Represents around $\frac{1}{10}$ of a full block as shown in Figure 3.23.

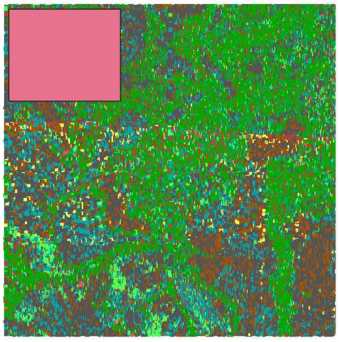


Figure 3.23: Toy dataset size comparison with a full block of 2km × 2km.

The green areas clearly represent the Canopy Cover as shown in Figure 3.24

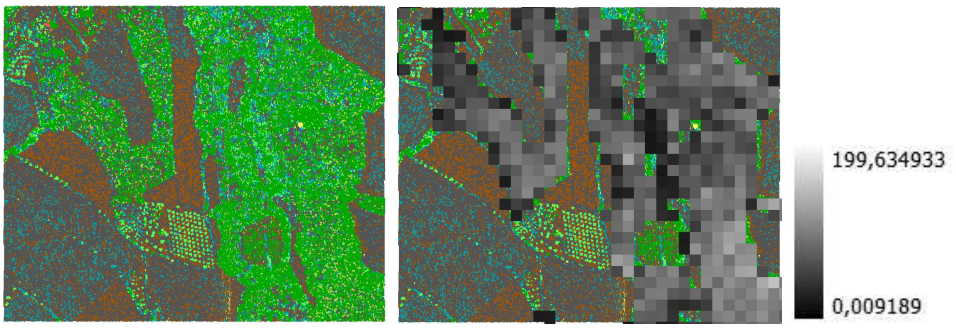


Figure 3.24: Toy dataset visualization and Canopy Cover comparison.

3. **Full block dataset**

- Purpose: An entire LiDAR block. Working with such blocks had significant relevance as it represented the minimum information according to how we had stored the data. Having insights on the processing time for this dataset would give us estimations of the processing times for the full dataset of Catalonia and subsets of it.
- Resolution: 2km × 2km. The Canopy Cover on that area surrounds ground areas. The block also has a lot of outlayers (probably birds

flying), buildings, cables and all the types of vegetation as shown in Figure 3.25, so it was a good candidate to start working with full blocks and checking Machine Learning Model (MLM)'s performance.

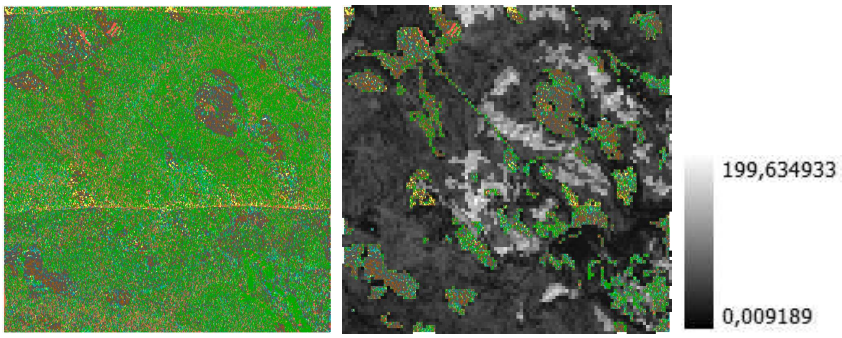


Figure 3.25: Full block visualization and Canopy Cover comparison.

4. **Nonzerogt dataset**

- Purpose: A region without zero valued canopy cover as shown in Figure 3.27. At the point it was created we were having trouble with a lot of zero valued canopy cover registers that got classified at high levels of canopy cover. To test the performance of our model in case this issue was sorted, we created this dataset that did not contain the problematic value.
- Resolution: 600m × 600m. Represents around $\frac{1}{10}$ of a full block as shown in Figure 3.26.

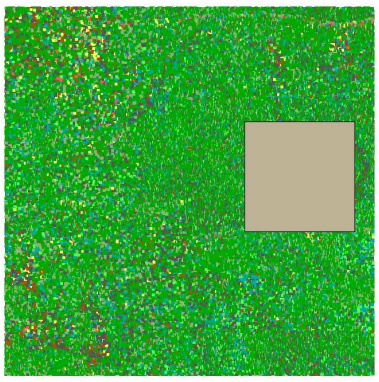


Figure 3.26: Nonzerogt dataset size comparison with a full block of 2km × 2km.

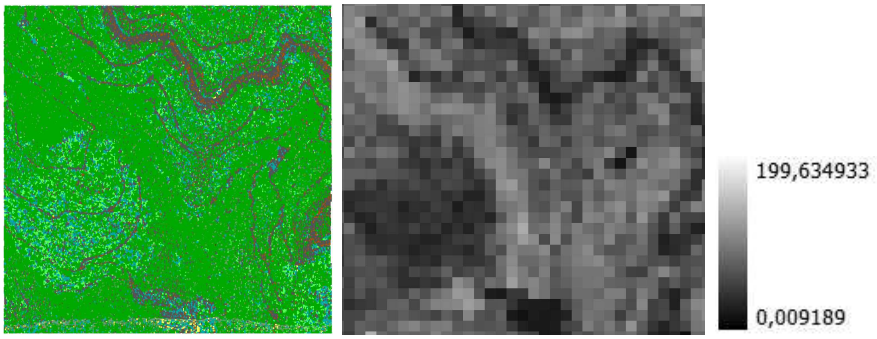


Figure 3.27: Nonzerogt block visualization and Canopy Cover comparison.

5. **Diverse10 dataset**

- Purpose: A set of 10 different blocks from all around Catalonia. The main purpose of this dataset was to chose each block from a different geography and vegetation type of Catalonia as seen in Figure 3.28. This dataset would contain geographies with buildings, rural zones, sparsely vegetated esplanades, mountains with few vegetation and areas with rivers and lakes. By training a model with that data we would theoretically be able to yield good results on anywhere in Catalonia.

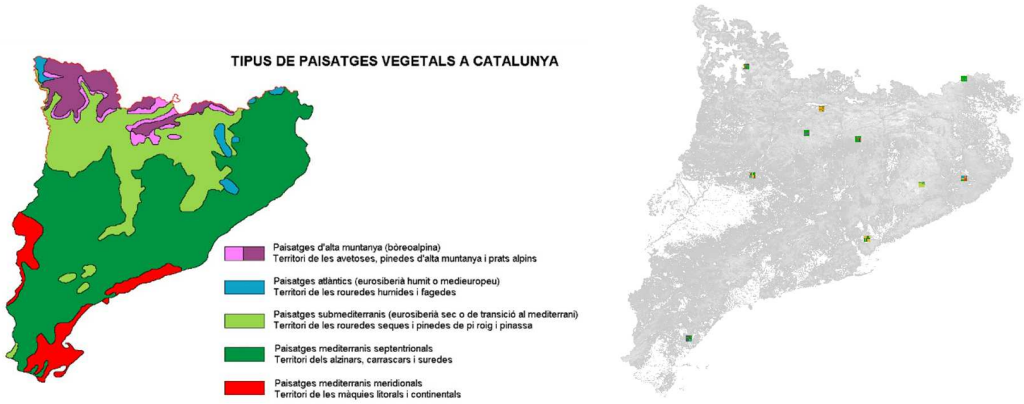


Figure 3.28: Catalonia's vegetation landscapes (left) and block distribution of the Diverse 10 dataset (right).

- Resolution: 10 blocks of 2km $\times$ 2km each.

6. **NextToBlock dataset**

   - Purpose: A block next to one belonging to the Diverse10 dataset. When we trained our model with the Diverse10 dataset and tested it on a random block from Catalonia, we noticed that the accuracy was extremely low. We concluded that it was because the diverse10 dataset did not have enough geographical information so the model could not extrapolate the information extracted from Diverse10 to the rest of Catalonia. For testing that hypothesis we gathered a block next to one of the Diverse10 as shown in Figure 3.29, as it would have a similar geography as the ones used for training the model. This block was then only used for testing purposes.
   - Resolution: 2km × 2km.

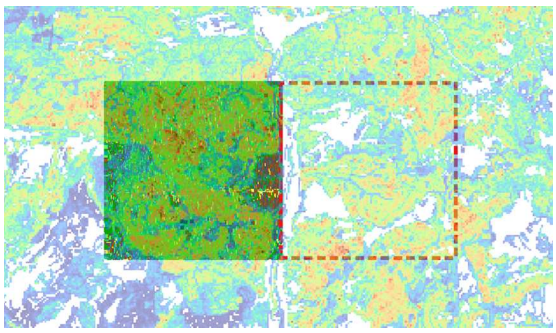


Figure 3.29: Block belonging to Diverse10 (left) and NextToBlock dataset (right).

3.5 Feature Engineering

Feature engineering is the process of selecting, manipulating and transforming raw data into features that can be used in supervised learning [35]. In this case most of the feature engineering involved taking aggregates of all the LiDAR pulses inside the same 20×20 m area that would characterize the zone and thus be the predictors of the model.

On this section I will explain the process behind the creation of the new features.

3.5.1 Height

Each LiDAR point is spatially located using three coordinates: x, y and z, the later reffering to the heigh of the point. However, it does not correspond to the distance between the ground and the measurement but between the geoid EGM08D595 and the measurement in the gravity direction (orthometric heights) [23]. As a consequence, the z value contains the height information but coded. This problem is often solved in the literature by creating a Digital Elevation Model (DEM) which is a representation of the bare ground (bare earth) topographic surface of the Earth excluding trees, buildings, and any other surface objects [53]. Knowing the height of the ground is then easy to calculate the height of each point.

There is not an established, unified and best way of calculating the DEM. Some algorithms determine the elevation of each grid cell by calculating the average of all LiDAR points classified as terrain within the cell [44]. Another method is Inverse Distance Weighting (IDW) interpolation method. Interpolation is the estimation of an unknown points using known points. The unknown points can be estimated through a radius or using a fixed number of known points surrounding the unknown point. The IDW is just a mathematical function for giving higher weights to known points closer to the unknown point than those far away. The closest points to the unknown point will have a greater say as to what value will be assigned to it. [46]. An additional method is through Triangular Irregular Network (TIN) gridding method. Gridding is the process of assigning a value to every pixel where there is a point. This value may be a min, max, mean or a mathematical function of all the points. The TINs are created by connecting the derived points (vertices) through a network of edges to form a network of triangles.

ICGC provides a DEM for all the Catalonia region with a resolution of 5×5m [25]. However, if we used that information it would make the pipeline very dependent on external data, as applying the pipeline to another area outside of Catalonia would require to have a DEM of the same resolution of the zone. Another downside of this approach is that the DEM is not usually updated and,

although the terrain does not usually change that much, a small change can be significative as estimating heights as accurate as possible is mandatory for having a good precision in canopy cover prediction.

In our case, we came up with a method for estimating the DEM directly from LiDAR data that consists on establishing the ground's height of a square region as the lowest registered LiDAR pulse. This method is based on the assumption that the LiDAR pulses hit the ground and thus the lowest registered pulse will be the one hitting the ground. It has the advantage that it only depends on LiDAR data, which will always be available. The downside is that there is a parameter that needs to be tuned, which is the resolution of the square region: if the resolution is too high, the DEM will have a large error on sloped areas, while if the resolution is too low, there might not be enough LiDAR pulses to estimate correctly the ground. Luckily, the LiDAR given on the real world case will have a much higer resolution (180 times more), and thus we will be able to estimate the height much more precisely.
For finding the optimal value in our case we used a grid and compared the estimations with the groundtruth (DEM from ICGC), and found out that establishing a square of 6×6 m yielded the best results. Figure 3.30 shows the predicted against the true values for the height of different tiles on a block, with a R-score of 0.97788.

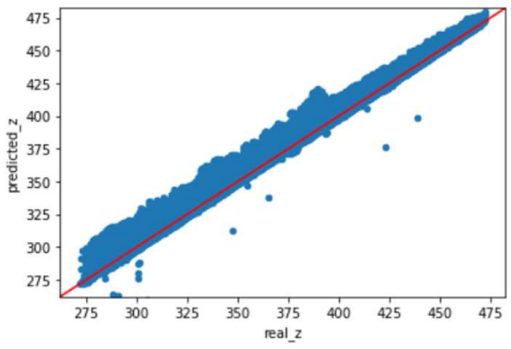


Figure 3.30: Real against prediction of height on the DEM with a R-score of 0.97788.

On a further analysis, Figure 3.31 shows the classification of the points that the algorithm used for establishing the height of a (different) tile.

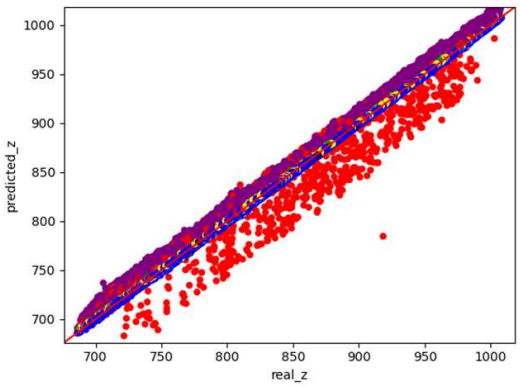


Figure 3.31: Classification of the points that the algorithm used for establishing the height of a tile. Blue: ground, Red: low point ($>$ 1m lower than neighbouring points), Purple: high vegetation, Black: not classified but suspicious of being ground.

The blue points do a great job at estimating the ground, as they are classified as ground itself. However, some tiles do not contain points classified as ground and other types of points are used. Low points (red) seem to highly underestimate the real height of the surface and thus including a lot of error into the model. High vegetation points on the other hand tend to slightly overestimate the real height and thus again include error into the model. This analysis can provide insights on how to base our height estimation in the future, keeping out the points classified as "low points" and trying to use only the ground ones.

We are aware that this method has some flaws and is based on some assumptions that are often not met, but when we designed it we were not aware of the literature on this topic and thus we are leaving this point as a possible significant improvement of the project.

3.5.2 Seasons

The LiDAR sensor will register different inputs on a same zone depending on the season when the measures were taken. On spring, trees will be populated with leafs which will result in a higher area where the lase can impact and thus more returns. On the other side, on autumn trees are less populated and then the canopy will have less area to be impacted on. This is the main reason behind monitoring in which season were the LiDAR data measured, as we considered relevant information that the model would want to take into consideration when predicting some atributes from LiDAR data. Although the perfect scenario would be to have the measurements all in the same season, it is not a realistic scenario as Catalonia's area of more than 32.000 km^2 can not be flight on reasonable time by an helicopter. As a result, the measurements were taken in different month between 2016 and 2017 as shown in Figure 3.32.

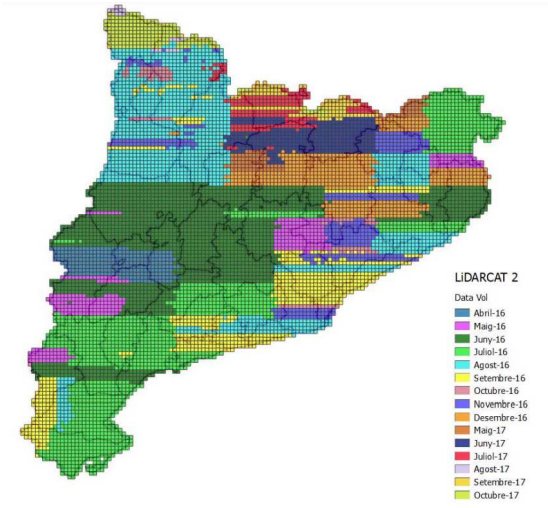


Figure 3.32: LiDAR flying dates of Catalonia per areas.

For calculating the season, each LiDAR point contains a feature called gps time that contains the time at which the pulse was registered and thus the season of the date can be inferred from that value.

3.5.3 Tree Tops

Tree tops is a simple and quick algorithm that estimates the amount of tree tops inside a 20×20 tile. The main motivation behind this number is that it should be proportional to the canopy cover (more tree tops would mean more canopy cover). This feature depends on a parameter called “radius". By changing this parameter multiple versions of this feature can be added to the model. The algorithm chooses the highest point on the tile classified as high or medium vegetation that is not marked as belonging to a tree top and sets it as a tree top. Afterwards, it considers from the same tree top all the neighbouring points inside the circle with radius set as parameter. Then, the algorithm repeats the same iteration until it cannot build another tree top and counts the number of them as shown in Figure 3.33.

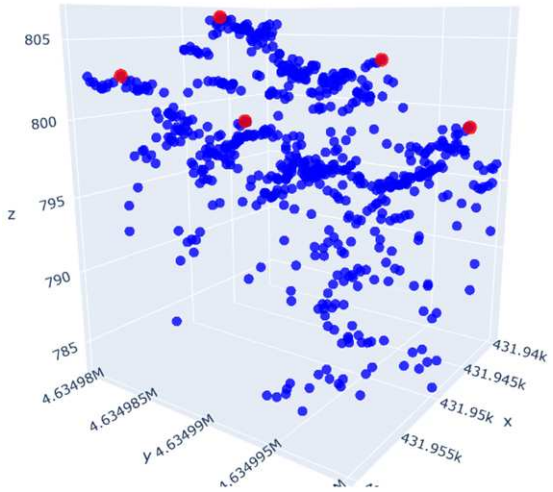


Figure 3.33: Tree tops found by the algorithm inside a 20×20m tile.

On a first instance, we planed to use a KMeans algorithm to build clusters of points and classifying them into either belonging to the canopy or the ground using the height of the cluster’s centroid under the assumption that the tree log would help separating the canopy from the ground and thus creating clusters. Then, having the LiDAR point classified as canopy or ground, we could calculate the number of tree tops that each tile has as the number of centroids above a certain height.
Another advantage of this method is that we could also consider the LiDAR

points inside a same canopy cover cluster as a tree's canopy and calculate the area of the tile that they represent, obtaining an estimation of the canopy cover. However, this method had a big disadvantage, as we had to find the optimal number of clusters, which involved trying multiple parameter values in real time and resulting in a high computational complexity that would drastically increase with more point density and directly interfered with one of the main goals of the project: to keep a minimum processing time.

Figure 3.34 shows the LiDAR points in a 20×20 area classified into three different clusters: high height canopy, medium height canopy and ground. In the end we opted to use a less-costly algorithm, which is the described above.

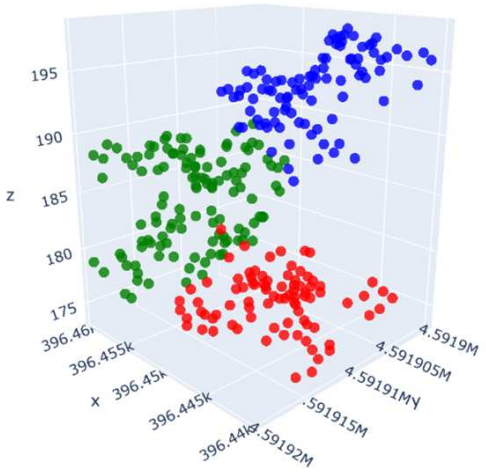


Figure 3.34: LiDAR data in a 20×20 tile classified into three different clusters.

3.5.4 LiDAR aggregates

We define as LiDAR aggregates simple features calculated only using LiDAR data that give information of a 20×20m tile by applying a function to the points inside that tile. Some of this features can translate into multiple columns in the DataFrame. For some features, we can even generate an unlimited amount of columns by varying a parameter. In this case, in order to keep a reasonable

computational cost, we only generated a few of them. The main idea is to build a large feature space consisting of a combination of simple features that together would add an important amount of information, and let the model decide which features are the most important for yielding a good prediction. Those features will then be the ones used to feed the final model with all the training data, and add up to form the bulk of the model, since we can generate a high amount of them at a low cost.

The aggregate features are:

1. **Number of points features**. For every type of LiDAR class (along with "not classified"), we generate a feature (a column) that calculates the percentage of LiDAR points inside the tile that belong to that class. All the different classes can be found in table 3.1.

2. **Mean features**. The mean of every numeric LiDAR attribute, such as intensity or scan angle. Attributes such as the x and y coordinates are not included. It can inform the model about the general behavior of the variables.

3. **Percentile features**. Explains the distribution of the data using 10% percentiles from 0 to 90%. Proved to be one of the most important features. It uses the same LiDAR attributes as the mean features.

4. **Standard deviation features**. Similar to the mean feature, the standard deviation is calculated. Explaining how sparse is the data can help understand its future behavior.

5. **Max difference features**. For every numerical LiDAR attribute (except x and y), the difference between the maximum and minimum value is calculated to help the model gaining a better understanding of the ranges of our data.

6. **Threshold percentage features**. Returns the relative amount of data points above a certain threshold. Only used on height, return number and intensity, each one with its custom thresholds according to its magnitude.

Chapter 4

Methodology

In this section I will outline the procedures and techniques used to conduct the research. First, I will introduce the different models that we worked with, detailing their strengths and complexity. Afterwards I will enumerate the metrics used for evaluating the performance of the different models and to rigorously comparing them. Following that step, I will write about the process of imputation and normalization used during the project as an important part of the preprocessing steps. Then I will showcase the training workflow that was followed in order to reach a final product, which will include a feature selection step and the hyperparameter tuning. Afterwards, I will introduce the use of the cluster along with the reasons why it was required, specifications, the limitations that we found, how we tried to solve them and the different tests that we performed on the cluster to study the scalability of our dataset. Finally, I will present the programmed software that will serve as the final product for the client, which will implement the prediction pipeline.

4.1 Models

As canopy cover can be estimated from LiDAR data in different ways, we built a variety of models which were getting more complex as the amount of available data became bigger. On this section I will outline those models, their strengths, weaknesses and where we used them.

1. **Height based model**: This model consists on estimating the canopy cover as the percentage of LiDAR points that are above a certain height threshold. This threshold is set as a parameter and can be optimized using a grid and comparing the results to a groundtruth. For this model a DEM is needed in order to find the real heights of the points. This first model was used in the smallest datasets (tiny, toy, nonzerogt and fullblock) to assess the viability of this kind of techniques and was crucial for understanding which information we should feed the Machine Learning model with.

2. **Vegetation based model**: This model consists on estimating the canopy cover as the percentage of LiDAR points that are classified as high vegetation (we also considered using high and medium vegetation but using only high yielded better results) under the assumption that high vegetation belongs to the tree's canopy. This second model was used in the smallest datasets (tiny, toy, nonzerogt and fullblock) to assess the viability of this kind of techniques and was crucial for understanding which information we should feed the Machine Learning model with. As a more advanced approach, instead of calculating the percentage of points that belonged to vegetation from the raw data, we performed some preprocessing, approximating each point inside the tile to the nearest integer x and y coordinate, and then only keeping the point that had the highest z on each pair of coordinates. Finally, we calculated the percentage of those remaining points that belonged to vegetation and it showed to increase the accuracy with respect to the baseline model.

3. **Machine Learning based model**: This approach consists on gathering as much information as possible about a 20×20m tile from many sources (not only LiDAR data) and using this information to train a Machine Learning model that will learn the underlying patterns of the data and how to use them to predict the canopy cover.
   Although it is the most capable model, it has some downsides. First, it requires the data to be preprocessed (data reading, data preprocessing, feature extraction, normalization, imputation, data merging, ...), which adds more execution time to the software and demands modelling decisions. Luckily,

we can drastically reduce this execution time by reducing the number of features using a feature selection step. Another weakness is that it requires hyperparameter search in order to optimize the model's performance. Furthermore, the model is prone to overfit or underfit if the training process is not done properly, so a correct assessment of the this phase is mandatory. Finally, this model requires a previous training phase to learn all the hidden patterns of the data and being able to extrapolate those to the new incoming data.

We assessed the performance of different Machine Learning Models (MLM) such as but not restricted to Random Forest, Suport Vector Machine or LightGBM over many datasets of this project. After comparing them and researching over different scientific articles that compared their performance and speed, we decided to use XGBoost. For instance, in Figure 4.1 the performance of random forest is assessed against XGBoost, the later showing a superiority. Furthermore, XGBoost took 5 seconds to train the model, while Random Forest took 87 seconds.

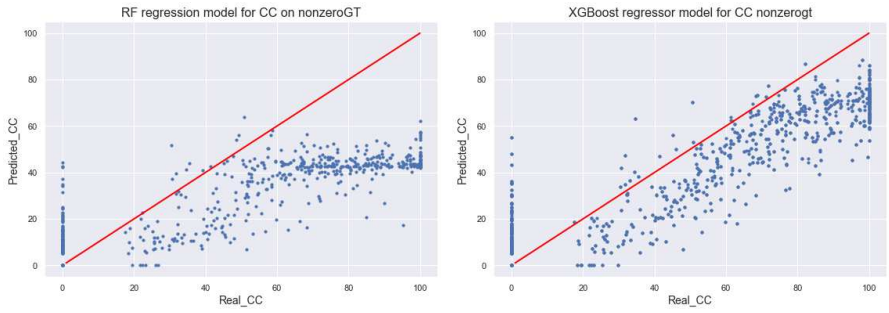


Figure 4.1: Random Forest versus XGBoost model comparison on the Nonzerogt dataset. The Random Forest's metrics (R^2 of 0.543 and MAE of 18.6) are beaten by XGBoost metrics (R^2 of 0.816 and MAE of 10.97).

XGBoost [5] is a gradient boosting algorithm designed for speed and performance. It has recently been dominating the applied machine learning and Kaggle competitions for structured or tabular data. It is based on the gradient boosting principles with some additions. As a scalable machine learning

system for tree boosting, it incorporates multiple improvements in terms of computation speed and performance. These innovations include: a novel tree learning algorithm for handling sparse data, a theoretically justified weighted quantile sketch procedure to handle instance weights in approximate tree learning, parallel and distributed computing to make learning faster which enables quicker model exploration and out-of-core computation. Most of these have never been explored on other parallel tree boosting algorithms, which gives XGBoost the upper hand. [8]

Although the first two models had a reasonable performance, the third model had a much bigger predictive power, so all our efforts were dedicate on training and improving that model. As a consequence, during this report, the mentioned techniques are dedicated to the Machine Learning based model unless mentioned.

4.2 Metrics

Metrics are an important aspect of the training and evaluation process, as they will be used to guide several key aspects of model development and assessment. Firstly, they facilitate the model's learning by minimizing the loss function. Additionally, they help in preventing overfitting and underfitting by comparing the metrics of the training set against those of the validation set. Metrics are also crucial in feature selection, enabling the comparison of models trained with different subsets of features to identify the most effective ones. Furthermore, they aid in selecting the best hyperparameters by retaining the subset that optimizes the metrics. Finally, metrics serve as the primary means of evaluating the overall performance of the model, providing a clear indication of its effectiveness.

In our case we have a regression problem, where we want to predict a numerical feature (Canopy Cover) that ranges from 0 to 100(%). Then, we will use R^2 score (coefficient of determination) as the main performance monitoring metric, as it shows how well a regression model (independent variable) predicts the outcome of observed data (dependent variable), and also because it is the most used one in the literature. R-Squared is a statistical measure used to determine the

proportion of variance in a dependent variable that can be predicted or explained by an independent variable. R-Squared values range from 0 to 1. An R-Squared value of 0 means that the model explains or predicts 0% of the relationship between the dependent and independent variables. A value of 1 indicates that the model predicts 100% of the relationship, and a value of 0.5 indicates that the model predicts 50%, and so on [12].

R^2 is calculated using the following formula:

$$R^2 = 1 - \frac{RSS}{TSS}$$

$$RSS = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

$$TSS = \sum_{i=0}^{n} (y_i - \overline{y})^2$$

Where n is the number of features, y_i is the expected output for the sample i, $\hat{y}_i$ is the predicted output for the sample i and $\overline{y}$ is the mean value of all the expected outputs.

Although R^2 is a good measure of how well is our model doing, it does not contain any real information on which error is our model expected to do. Thus, additionally to the R^2 metric, we also have decided to keep a record of the Mean Absolute Error (MAE) of the model. The MAE is defined as the average variance between the expected values in the dataset and the predicted values in the same dataset [2]. It can be interpreted as the expected error that the model will have on its predictions and thus gives important information on which errors to expect while using the model in production.

The MAE is calculated using the following formula:

$$MAE = \frac{1}{n} \sum_{i=0}^{n} |y_i - \hat{y}_i|$$

While training the model, the loss function to be optimized will be the Root Mean Squared Error (RMSE). RMSE squares the errors, which gives more weight to larger mistakes. In our context, larger errors are more costly and thus it is important to minimize them, which is the main reason why we used RMSE instead of MAE [2]. RMSE can be calculated using the following formula:

$$RMSE = \sqrt{\sum_{i=0}^{n} \frac{(y_i - \hat{y}_i)^2}{n}}$$

4.3 Imputation

Data imputation is a method for retaining the majority of the dataset's data and information by substituting missing data with a different value. These methods are employed because it would be impractical to remove data from a dataset each time. Additionally, doing so would substantially reduce the dataset's size, raising questions about bias and impairing analysis [48]. In out case, although we did not find any NaN values on the dataframes, we still added a SimpleImputer [49] to our script in case in a future use case was needed. In our case, the SimpleImputer replaces the missing values of a column with a mean of the existing values for that column.

4.4 Normalization

Normalization is an essential step in the preprocessing of data for machine learning models, and it is a feature scaling technique. Normalization is especially crucial for data manipulation, scaling down, or up the range of data before it is utilized for subsequent stages in the fields of soft computing, cloud computing, etc [17].
For this project we chose to use StandardScaler[50] normalization to transform features into a similar scale by ensuring each column has mean 0 and standard deviation of 1, as it has shown to improve the performance and training stability of the model [20].

4.5 Training Workflow

In this section I will explain the workflow followed for predicting a biophysical variable (such as Canopy Cover), detailing the different scripts at each step of the process along with their purpose and functioning.

4.5.1 Feature Selection

Feature selection is the process of isolating the most consistent, non-redundant, and relevant features to use in model construction [21].

Performing feature selection is important for many reasons. In computational terms, with less features the code is naturally faster. This, in turn, allows us to train with more data, which is mandatory in our case, as we wanted to train our model using all of Catalonia data. On the other hand, with less features we can interpret and explain what the model is doing better, as well as avoiding overfitting easier.

The process for performing feature selection was done choosing a small portion of all the data (around 5% of Catalonia) as we had to calculate a significant amount of features.

4.5.1.1 Script's explanation

A visualization of the data processing methodology before proceeding with the feature selection is shown in Figure 4.2.

First, all the necessary libraries are imported and the constants such as number of cpu used or the number of blocks to be used are defined. Then the specified number of blocks are randomly subsampled from the Catalonia's database and then readed in parallel to ensure fast results. For each block, the reading function inputs LiDAR data from the .laz files using *laspy* library [52] and encodes the seasons to convert the gps time into a meaningful feature. Then, each block is preprocessed, removing outliers (provided that they do not add any important

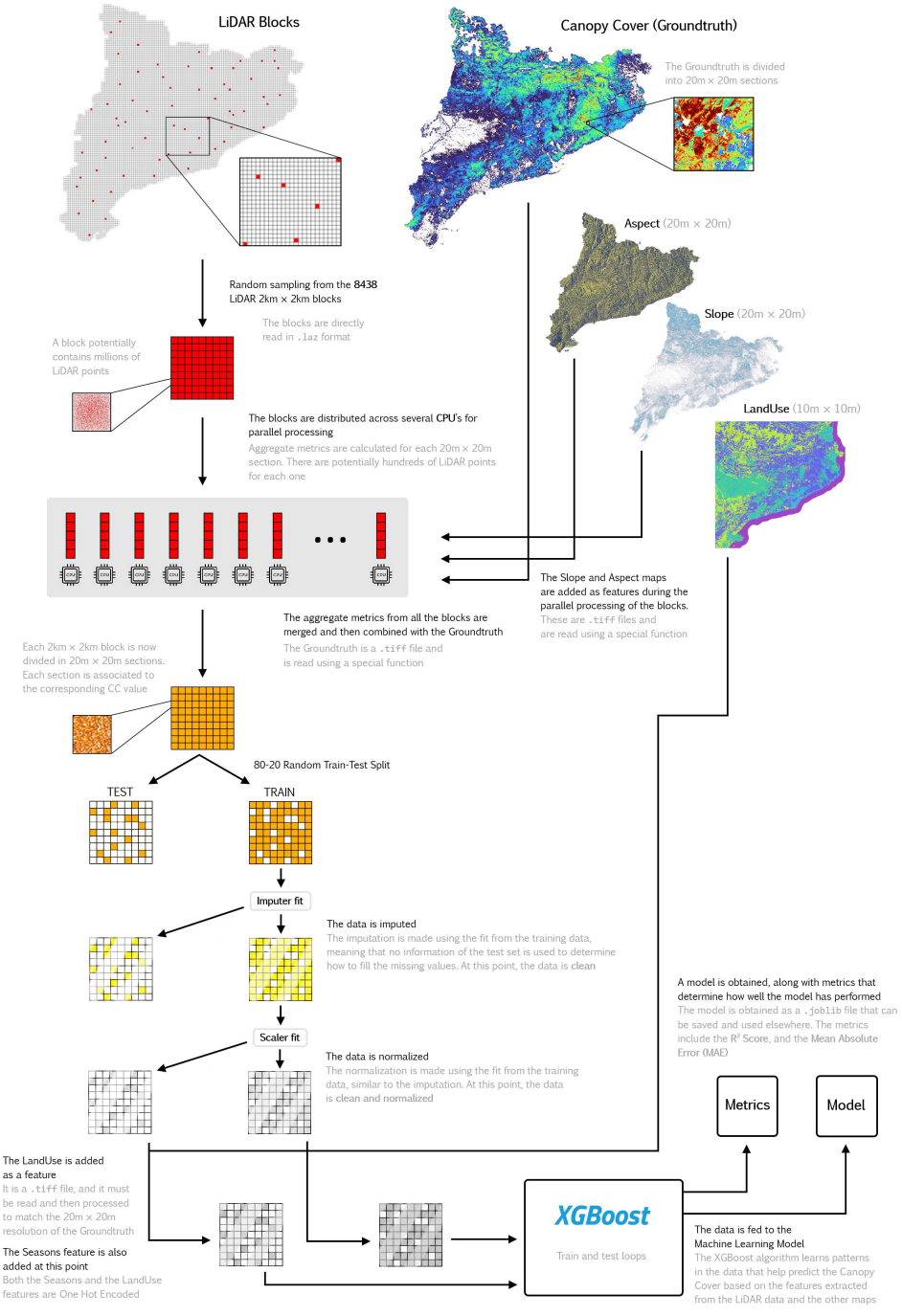


Figure 4.2: Methodology overview.

information as Figure 3.6 shows), adding a feature to each point that labels which tile does it belong to, calculating the DEM and grouping the points from each tile to create the aggregate features (and the Tree Tops feature) in a parallelized manner.

Once that preprocessing is done, now each block is not characterized by a set of LiDAR points but by the set of tiles that forms it and the aggregates that define the information from each tile. At this point each tile has around 140 features.

After that, the groundtruth data is read from a *.tiff* file using the *rasterio* library [26] and, in case of canopy Cover, the values are limited to a maximum value of 100. Following that step, the groundtruth information is merged in a parallelized manner into each LiDAR dataframe, establishing the value to be estimated from each tile. The same exact process is applied to *slope* and *aspect* tiffs, adding the aspect and slope values to each tile of each block. Then, all outliers from the aspect and slope data (with value of -9999) are replaced by NaN as they will be imputed in the future.

At this point, we have an array of dataframes, each representing a different block. Each row of each dataframe represents a different tile of 20×20m inside a block. Then, it is important to randomly split the data in train and test, as we will be using 80% of our dataset to train a set of models with a different number of features and the rest for comparing their performance. The blocks at each split are concatenated into a combined dataframe so it is easier to manipulate the data. The target variable (Canopy Cover) is stored and removed from the dataframes.

Then, data is imputed and normalized according to the specifications made on the previous section. Normalization and Imputation are trained on the training split and then used as well for the testing split. Following normalization, the script reads and processes the land use in order to establish the x and y coordinates to match the 20 × 20 m resolution (as the initial resolution is of 10×10m). Landuse and season features are added to the model using a One Hot Encoding

[3], as it contains categorical data. The reason behind doing it after splitting the data is due to the fact that we wanted to avoid the encoded features from being normalized.

Finally, the feature selection process begins. A simple model is trained with all the features and the importance of each feature is calculated. Given a certain importance threshold, all the features that are below that threshold are discarded and a new model is trained with only the kept features. This process is repeated lowering the threshold value at each step until only one feature is remaining. Then the performance of all the trained models on unseen data is tested plotting R^2 score to decide the best features to be used.

4.5.2 Hyperparameter Tuning

Hyperparameter tuning is the process of selecting the optimal values for a machine learning model's hyperparameters. By training a model with existing data, we can fit the model parameters. However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn [18]. The importance of hyperparameter tuning lies in its ability to improve the model's performance and generalizability. By fine-tuning these parameters, the model can be adjusted to better fit the specific dataset and problem, potentially leading to more accurate predictions.

It is highly advised to use at this point a large subset of the total available blocks as speed should not be a problem due to the filtering done in the previous step. Then the model is trained with the best hyperparameters found and using 80% of Catalonia and tested using the remaining 20%. The trained model and other necessary instances will be saved to be used "a posteriori".

XGBoost hyperparameters can be divided in 3 categories: general parameters, booster parameters and learning task parameters (there is an extra category for

xgboost's implementation in command line, but it is not relevant for this study). The focus will be placed on the booster and learning task parameters, as changing general parameters can lead to a paradigm shift. The general parameters will be kept at their default values, except for nthreads, which is recommended to be adjusted based on the number of CPUs available on the machine [8].

1. **Booster parameters**. Although 2 types of boosters exist, the attention will only be directed towards the tree booster because of its overall better performance. Those parameters control the construction and complexity of the decision trees. The list of hyperparameters that are being tuned is:

   (a) min_child_weight: Determines the minimum sum of instance weight needed in a child node. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can lead to under-fitting. Searching space: [1, 20, 50, 100]

   (b) max_depth: Controls the maximum depth of a tree. Deeper trees can model more complex patterns, but they can also overfit and become less interpretable. Searching space: [0, 5, 25, 50, 100]

   (c) subsample: Controls the fraction of observations to be randomly sampled for each tree. Lower values make the algorithm more conservative and prevents overfitting but too high values might lead to under-fitting. Searching space: [0.1, 0.5, 0.75, 1]

   (d) colsample_bytree: Controls the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed. Searching space: [0.1, 0.5, 0.75, 1]

2. **Learning task parameters**. Those will control the overall behavior of the model and the learning process. Those include:

   (a) learning_rate: It shrinks the feature weights to reach the optimum faster. After each boosting step, it can be interpreted as the probability of a correct classification. Too small values can slow down the training process. Searching space: [0.01, 0.1, 0.5, 1]

(b) gamma: This is a regularization parameter. The larger gamma is, the more conservative the algorithm will be. Higher values reduce the complexity of the model and help prevent overfitting. Searching space: [0, 0.5, 2, 5]

(c) reg_lambda: L2 regularization term on weights (xgboost implementation calls this lambda). Searching space: [0, 0.5, 2, 5, 10, 50, 100]

(d) reg_alpha: L1 regularization term on weights. This can help control overfitting by adding a penalty equivalent to the absolute value of the magnitude of coefficients. Searching space: [0, 0.5, 2, 5, 10, 50, 100]

(e) n_estimators: Number of gradient boosted trees. Searching space: [500, 1000, 2500, 5000, 10000]

4.5.2.1 Script's explanation

The baseline of the script is the same as the one used for Feature Selection. Data is readed and processed the same way using the same functions. However, when extracting the features of each tile, only the ones chosen from the Feature Selection process are calculated. After the imputation and normalization of the different sets, the hyperparameter tuning phase begins.

For iterating through the search space a Cross Validated Random Search is used, which is a technique used for hyperparameter optimization in machine learning models. It works by randomly selecting a combination of hyperparameters from a predefined distribution to train a model. The performance of the model is evaluated using 6-fold cross-validation, which it involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance [16].

This process is repeated a certain number of times. The best performing combination of hyperparameters is chosen as the optimal solution.

After finding the best hyperparameters, they are used to fit a model with 80% of Catalonia, and the same model is tested with the remaining 20% to evaluate its performance with the previously mentioned metrics. The trained model is then saved along with the Scaler and the Imputer and ready to use on the final product.

4.6 Execution

4.6.1 The cluster

Although working with a large amount of data provides the model with robustness and precision, it also requires large amounts of memory and execution time. For instance, Catalonia's LiDAR data weighted around 3TB, more than three times the amount of total space of our computers, and each tiff weighted some more gigabytes, which could not fit into a regular RAM device. All this limitations made mandatory to use a powerful computer.

Autonomous University of Barcelona's (UAB) Computer Architecture and Operative Systems (CAOS) department provided us with one of their clusters, the HPCA4SE. We were given access to a node with Intel Xeon E5-2620 architecture, 2 sockets, 6 cores and 24 CPUs.

With access to that cluster we were ready to take our executions and tests to another level. However, it also involved learning the basics of cloud computing and building scripts (as we were using python notebooks since then).

We managed to upload all 8434 Catalonia's compressed LiDAR files in 5 hours and unzipped them in 30 minutes. The total amount of compressed data weighted 213 GB.

4.6.2 Script limitations

4.6.2.1 Execution Time Issues

The main barrier we found on this project was the execution time. Having that many features and blocks to be processed added a lot of complexity to the model, resulting in large execution times that depended not only on the amount of features and blocks but also on the sizes of each block. Figure 4.3 shows that the processing time seems to have a linear relationship with block size, which is in fact good news.

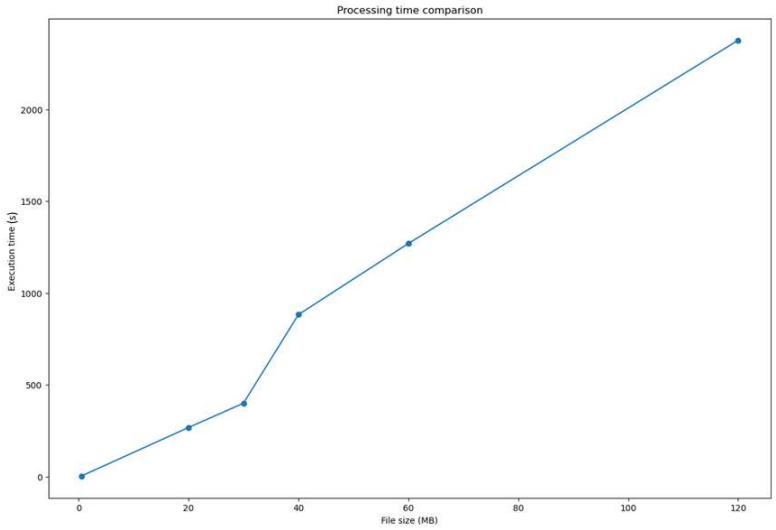


Figure 4.3: Processing time evolution depending on the size of the LiDAR block.

The largest blocks will take around 41 minutes to be processed, while the smallest ones can take less than one minute. However, those are only outliers. We have calculated that the mean processing time among blocks is of 7-9 minutes (in sequential). This means that processing the 8434 blocks of Catalonia would take between 41 and 53 days. This is both unsustainable and unnecessary, as there are many irrelevant features that are being calculated. If we want to offer a fast product we cannot afford to use 8 minutes on processing each block.

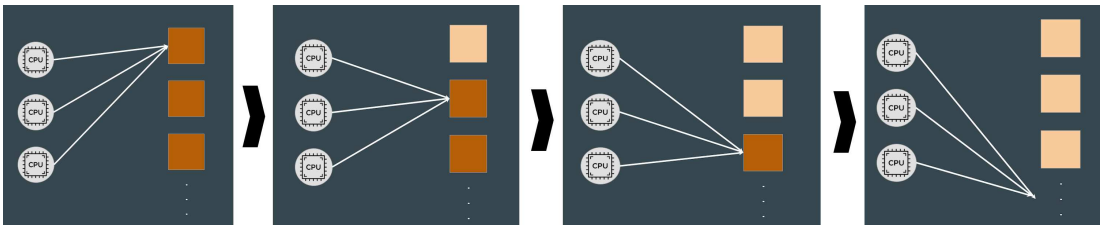
Four main optimizations have been done in order to reduce the processing time:

- **Reducing the number of features**: At this point the model was working with 152 features, most of which were highly correlated. Although the features were simple to calculate, for each block we had to calculate the 152 features for each one of the 10.000 20×20m tiles. If we managed to reduce that number we would significantly lower the processing time. More on this in the Feature Selection section.

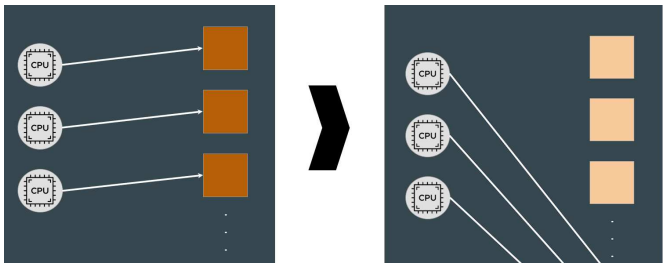
- **Parallelization**: In order to take full advantage of the cluster's computational power and the device that will be used by the firefighters, we parallelized the software to make sure that it used all the available CPUs. The main part that we parallelized (as it was the most computational expensive) was the aggregation of LiDAR points inside each 20×20m tile, as each block is independent from the others and thus could be calculated in an isolated manner.
  We tried many python parallelization libraries before getting to the optimal one.
  First, we tried Parallel-Pandas, a library that parallelized the panda's apply function [33]. This library is very useful when having few very costly apply functions. However, our case is the opposite one, we have a lot of simple built-in functions which were indeed faster than paralleling the apply.
  Another library we tried was Dask [7]. Dask is an open-source Python library for parallel computing which scales Python code from multi-core local machines to large distributed clusters in the cloud [57]. However, when we tried it we found that it did not optimize that much small operations, as for applying 100.000 means the execution time increased from 4.3 seconds (sequential) to 44.7 seconds ("parallel"). When tried on 1.000.000.000 means, the execution time was reduced from 80 seconds (sequential) to 57 seconds (parallel). Given the small optimization that supposed, we decided that the library would not be the best fit for our case.
  Finally, we found the multiprocessing library. Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the Global Interpreter Lock by using sub-

processes instead of threads [9].

The parallelization design used also evolved through different stages. First, all CPUs were invested in one block in order to process it as fast as possible. Although this approach had a low memory usage, as only one block is loaded in memory at the time, the time that the data took to transfer to each CPU's memory was even higher than the time the CPU spent processing their chunk of the data, leading to very poor results (even when comparing them with sequential approach). As Figure 4.4 shows, we improved the parallelization by processing one block at each CPU.



(a) First approach: All CPUs focus on just one block. When the processing is finished, they move to the following block.



(b) Second approach: Each CPU focuses on one block. When the processing is finished, they move to the next set of blocks.

Figure 4.4: First parallelization improvement.

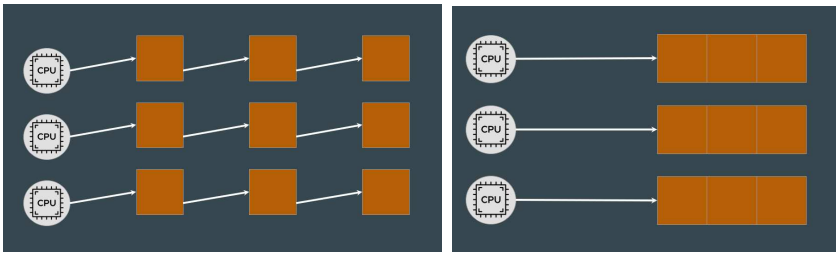
This way we highly increased the memory usage but gained a substantial amount of speed. However, this design had a flaw: each CPU had to wait at each round until all others finished processing their block before starting a new one. This was a problem, as for each round the execution time of

all blocks would be equal to the block that took the most to execute. In a mathematical form, for each given round i:

$$ET = \sum_{i=0}^{m-1} \max_{j=0..n-1} \{B_j^i\}$$

Where ET is the total processing time, B_j^i is the execution time of the block processed by CPU j on the round i, n is the number of CPUs and m is the number of blocks per CPU (note that the total number of blocks can be calculated using mn and note as well that we are considering a number of blocks multiple of CPUs to simplify the explanations). If all blocks on the same round had a similar size, this would not be a big problem. However, as shown in Figure 2, the sizes of blocks vary significantly.
On the one hand, in the rounds where there is an unusually large block there would be a delay in all the other block's processing, as all others CPUs would have to wait until the large block is finished processing, even though they would have finished their processing, wasting computational power. On the other hand, in the rounds where there is an unusually small block, the CPU that was processing that block would have to wait a long time until all other blocks from that round are finished processing, wasting computational power.

In order to reduce that side effect, as Figure 4.5 shows, we decided to assign all the blocks that will be processed by each CPU from the start, as that would prevent each CPU from waiting to all others at each round.



(a) Second approach: Each CPU focuses on one block. When the processing is finished, they move to the next set of blocks.

(b) Third approach: Each CPU focuses on a set of blocks such that all the blocks are assigned to a CPU from the begining.

Figure 4.5: Second parallelization improvement.

Although with this design CPU still have to wait when they end up all their blocks, the number of waiting times is reduced from m (each round) to 1 (only in the end). Following the same nomenclature as previously, now the total processing time is:

$$ET = \max_{j=0..n-1}\{\sum_{i=0}^{m-1} B_j^i\}$$

Note that this time is at least lower than the previous one:

$$\max_{j=0..n-1}\{\sum_{i=0}^{m-1} B_j^i\} \leq \sum_{i=0}^{m-1} \max_{j=0..n-1}\{B_j^i\}$$

This can be proven using the triangular inequality [62], which states that $||x + y|| \leq ||x|| + ||y||$ for every norm. This expression can be generalized to $||\sum_i f_i|| \leq \sum_i ||f_i||$ where f_i is a any value so we can define $B_j^i := f_i$. Now notice that in our case, $B_j^i \geq 0$, which lets us use the infinity norm which is in fact, the maximum function [59], leading to the expression $\max\{\sum_i B_j^i\} \leq \sum_i \max\{B_j^i\}$ QED.

To further optimize the code, we also parallelized the tiff merging process. When we have the dataframes where each row represents a 20×20m tile

with the LiDAR aggregates, we want to add the tiff information (slope, aspect, groundtruth and landuse). However, each of those maps represent the whole Catalonia and need to be merged into each dataframe, which involves a large amount of comparisons. To avoid investing too much time on that, we also parallelized this part using the same design strategy as in the previous paragraph.

The last part we parallelized was the XGBoost model, as it has an argument that allows us to train the model much faster by using multiple CPUs.

- **Vectorized operations:** NumPy is a Python library, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays[31]. NumPy implements vector operations that are backened with C and Fortran languages [60], which makes them extremely fast and efficient compared to regular Python operations, from 5 to 100 times faster [54]. Some of the operations of our code could actually be written as numpy vectorized operations. For example, setting all values greater than 100 on the groundtruth to 100 can be done easily with Python using a while loop and a conditional. However, using NumPy this operation can be done in a faster, more efficient and elegant way. More examples of operations are labeling each LiDAR point to its belonging 20×20m tile for calculating aggregates or grouping LiDAR points to its belonging 6×6m area for calculating heights.
- **Preprocessing before pandas merging function**: At one point we had one dataframe for each block containing tile aggregate information on each row and we wanted to add the information from tiffs maps. This tiff maps contain data about the whole Catalonian map, but we only needed the area where the block was processed, as directly merging the tiff with the block's dataframe would involve a large number of unnecessary comparisons. As a solution, we decided to first cut the tiff to the square area that represented the block, as we had that information from the minimum and maximum values of the dataframe's columns that contained the coordinates. This way when merging the number of comparisons were drastically reduced.

4.6.2.2 Memory Issues

The main issue with memory that we confronted was with the parallelization of the code, as having a big amount of data on each CPU drove the cluster out of RAM memory, creating a bottleneck on the script that would slow up the whole execution. To address this problem, we added some optimizations to the code as shown below:

- **Block reading**: Most of the tests did not use all the blocks but a subset of them. Instead of reading all LiDAR blocks and then randomly choosing which ones to process, we first choose which blocks will be processed assigning an index to each block and then only read those blocks chosen to be processed.

- **Data types**: When we readed the data, most datatypes were extremely precise. However, we did not need that much precision. Due to the limitations of the coordinates of the tiffs for example we knew that the coordinates would always fit into a *int32* datatype instead of a *int64*, reducing the memory usage to half. Furthermore, all datatypes for LiDAR points and aggregates were reviewed and changed according to a more suitable type. Another example of changing the types is in landuse, as the classes can only be from 1 to 10, which allows us to use only a byte to represent them.

- **Merging Tiffs**: One of the worst bottlenecks was while merging the tiff data to LiDAR data in parallel for each block. At this point we had one dataframe for each block containing tile aggregate information on each row and we wanted to add the information from tiffs maps. The issue comes when it has to be done in parallel, as this would involve creating as many copies of the full tiff (which weighted some gigabytes) as CPU used. A solution was to cut the tiff before sending it to process each block, as we did not need the full map, only the same square as where the block was. However, this operation would then not be parallelized so we decided not to implement this solution.

4.6.3 Tests done

Having the cluster allowed us to drastically scale the number of blocks used. In local we only managed to process 10 training blocks (diverse10 dataset) and use 1 for testing purposes (NextToBlock dataset).

To study how the execution time increased in relation to the number of blocks processed, we started incrementing our data to a total of 24, 48, 72, 96, 120 and 144 by using the cluster. Note that those were multiples of the number of CPU (24) to use all the computational power. Even though those numbers are large in comparison of the tests done on local, they still represented a small percentage of Catalonia (144 represents 1.7% of Catalonia), but would give us enough information to estimate how much would it take to process all the blocks. Then we performed Feature Selection using 5% of Catalonia (around 400 blocks). Afterwards, we reduced enough the number of features that we could process all Catalonia in a reasonable time, so we performed Hyperparameter Tunning with the 80% of Catalonia and then trained the final model on the same dataset, using the rest 20% for testing its performance.

4.7 Final Product

As this is a model that is supposed to be used on real cases, we decided to farther develop an automatized software that would allow the client to get the predictions from any desired block by reading the blocks, processing them, normalizing and imputing the data, reading the previously saved model and using it for the prediction.
The script not only is fully optimized but also allows the user to use multiple cores for parallelization.

The final product will be ran from the terminal, accepting the following parameters:

- input_blocks_path: path where the input block / s are stored.
- output_blocks_path: path where the output prediction will be stored.

- –output_type, -o: to select the format of the CC block prediction. The type outputs allowed are: .csv (default), .tiff and .asc.
- –n_cpu, -n: Number of CPU that will be used (default: -1 -> max cpu available).

Given the input path and the output type, the script will provide the CC prediction for each block which will be stored in the specified directory.

Chapter 5

Results

This chapter collects the results on all the different steps of the process, outlining the findings from the different models, evaluations of simple models, outcomes of feature selection and hyperparameter tuning, and comprehensive training results for the diverse datasets, along with interpretations.
For valorating the model's performance and results we will use the previously mentioned R^2 and MAE metrics, along with a scatter plot of predicted vs. actual values and in some cases the map with the predicted values next to the map with the expected values of the same area. The scatter plot of predicted vs. actual values is a graph that shows on the x-axis the actual values and on the y-axis the predicted ones, so the perfect accuracy would be for the points to fall on the diagonal line, which would be that all the predicted values matched the real values.

5.1 Simple Models

Simple models include all those models that do not use artificial intelligence for predicting Canopy Cover. Those models were just built for checking the predictive power of designs proposed by the current literature, so we decided not to invest all our efforts on that and thus the models were only used on smaller datasets provided that the MLM performed significantly better. The simple models included the height based model and the advanced approach on the vegetation

based model.

For the height based method, we did a grid and compared a list of thresholds on a full block as shown in Figure 5.1, with the best one being 8.

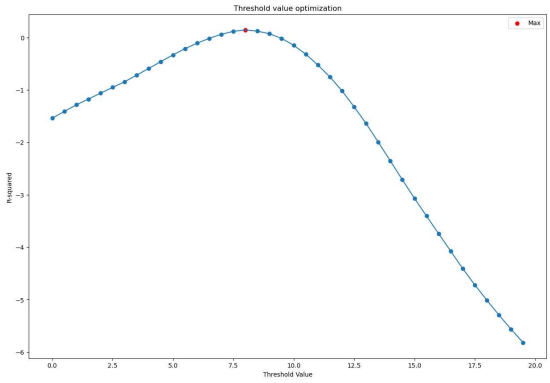


Figure 5.1: Threshold value optimization for the height based method.

The results of all the tests done are collected on the following table:

	Vegetation Model		**Height based Model**	
Datasets	R^2	MAE	R^2	MAE
Tiny	0.534	10.523	0.118	14.772
Toy	0.554	17.843	0.563	16.466
Fullblock	0.230	24.181	-1.50	42.18
NonzeroGt	0.346	15.652	0.144	17.872

As most of this datasets did not suppose a significant area and thus the results would not be rigorous, we decided not to include deeper analysis on those such as plotting the predictions or scatter plotting with the real values.

Across the evaluated datasets, the Vegetation Model consistently outperforms the Height-based Model in both R^2 and MAE metrics. This suggests that the classification of LiDAR points as high vegetation provides a more reliable basis for canopy cover estimation compared to a simple height threshold.

The Tiny Dataset results indicate that both models handle simpler, smaller datasets better, though the Vegetation Model is superior.

The Toy Dataset results suggest that both models can handle moderately sized datasets, with the Height-based Model slightly outperforming the Vegetation Model.

The Fullblock Dataset results highlight significant challenges for the Height-based Model, suggesting it may not be suitable for large and complex areas.

The NonzeroGt Dataset results show moderate performance for the Vegetation Model and poor performance for the Height-based Model, reflecting difficulties with more diverse canopy structures.

While the simple models provide a baseline, their moderate to weak performance specially on larger datasets highlights the need for more sophisticated approaches, such as machine learning models that can capture complex patterns and interactions in the data.

The variation in performance across datasets underscores the importance of dataset characteristics in model evaluation and the necessity for models that generalize well across different types of data.

In conclusion, the simple Vegetation-based model shows better performance than the Height-based model, but both have limitations, especially on more complex or varied datasets. This initial analysis corroborates that moving towards more advanced machine learning models is a promising direction and serve as an introduction to the more complex datasets and models.

5.2 Complex models

This approach consists on gathering as much information as possible about a 20×20m tile from many sources (not only LiDAR data) and using this information

to train a Machine Learning model that will learn the underlying patterns of the data and how to use them to predict the canopy cover. On this section we will be going through some datasets that we used to train and test our machine learning model, increasing their size and thus our model's performance.

5.2.1 Diverse 10 Dataset

The first dataset where we tried our machine learning model was the Diverse10, training it with 70% of the tiles and testing it with the remaining 30%. The model achieved significantly good results as shown in Figure 5.2, with a R^2 score of 0.951 and a MAE of 4.47.

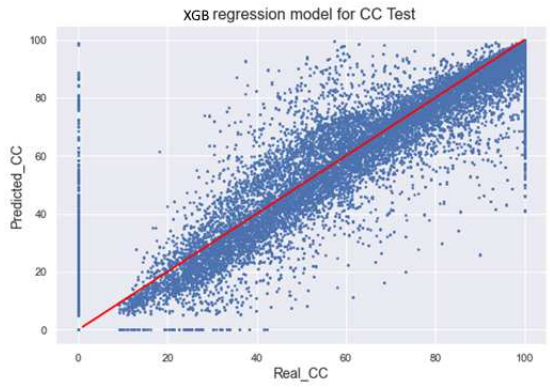


Figure 5.2: Scatter plot real values vs. predicted values of the Diverse10 dataset having tiles randomly sampled from diverse blocks as testing partition. The red line indicates optimal values.

The main idea behind Diverse10 dataset is that it contained distinct types of geography and thus training a model would hopefully mean that it could generalize to the rest of Catalonia. However, this was not the case, as when we tested the model on a random full block of Catalonia it reported a very low performance as shown in Figure 5.3 with a R^2 score of -1.87 and a MAE of 41.34.

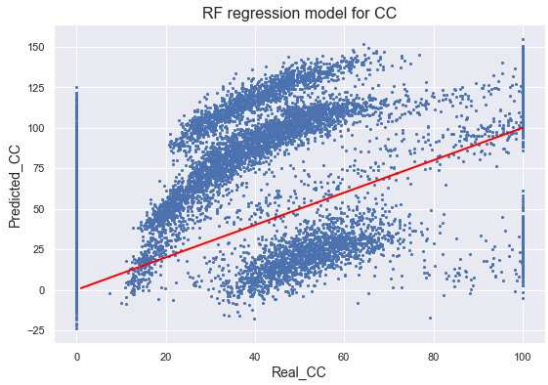


Figure 5.3: Scatter plot real values vs. predicted values of the Diverse10 dataset having a full block as the testing partition. The red line indicates optimal values.

Our hypothesis relied on the fact that Catalonia's geography was too complex to be modeled using only 10 blocks and thus we needed far more data to train our model. Although our model would not be able to accurately predict a random full block of Catalonia due to its differences with the geography of the ones used while training, we were confident that it could predict a block with a similar geography, and thus tested the model on a full block situated next to one of the samples belonging to Diverse10 dataset (the NextToBlock dataset).
The tests reported a R^2 score of 0.82 and a MAE of 10.79, and a scatter plot that contained a peculiar pattern as shown in Figure 5.4.

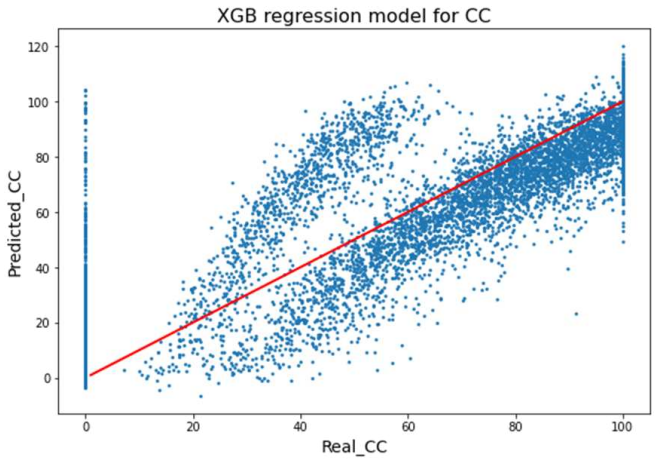


Figure 5.4: Scatter plot of real vs. predicted values using the Diverse10 dataset for training and NextToBlock dataset for testing.

We will dig deeper into this pattern in the discussion section, but the model yielded the desired results. Figure 5.5 shows the NextToBlock's real Canopy Cover compared with the prediction made by our model.

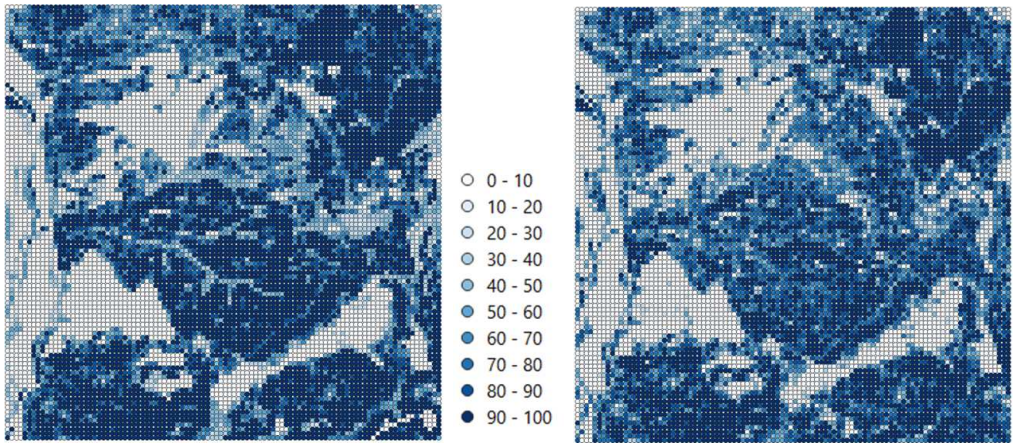


Figure 5.5: Canopy Cover map comparison between the real values (left) and predicted values (right).

Overall, the prediction achieves a good result on forecasting similar shapes as the real values, but fails on the more detailed parts, as when comparing both maps those seem o be blurred. However, our goal with this test was not to achieve

the best performance but to prove that with similar geography our model could yield good predictions, which we achieved.

5.2.2 Cluster Execution Results

In this subsection I will be presenting the different results obtained from training our model with a larger set of data, how the problem scaled and how parallelization impacted the execution time and memory usage.

The main problem of scaling our dataset to all Catalonia was the long processing time as shown in Figure 4.3, with an average processing time of around 8 minutes, which would result in 46.8 days.

After parallelizing our code with 24 CPUs, we recorded the growth in execution time in terms of the number of blocks processed, from 24 to 144. As Figure 5.6 shows, the improvement is impressive, saving up up to almost 15 hours on the last stage.

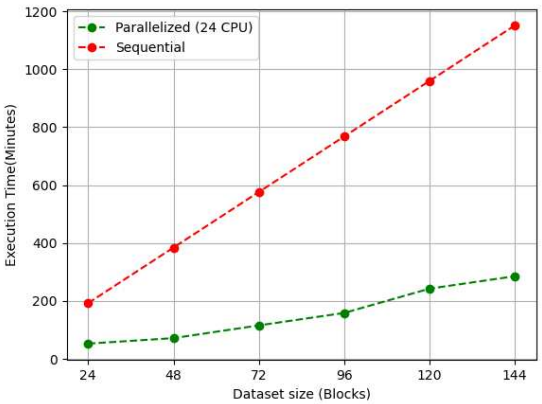


Figure 5.6: Script execution time comparison between an estimation of the sequential approach and the measured parallel results.

By parallelizing our code, we achieved to process each block in almost 2 minutes when we use 24 cores, which is a good optimization when compared to the 8 minutes that it takes in sequential. Another way of comparing how well is our

parallelization doing is by using Speedup and efficiency. In computer architecture, speedup is a number that measures the relative performance of two systems processing the same problem. More technically, it is the improvement in speed of execution of a task executed on two similar architectures with different resources [61]. Efficiency is defined to be the ratio of the speedup to the number of processing elements [27]. The Speedup can be calculated by dividing the time that a task takes to execute in sequential by the time that the same task takes to execute in parallel. In our case, from our data we have calculated a Speedup of 4.5 and an efficiency of 19%, which is a good achievement, as our program is 4.5 times faster using 24 CPUs.

By breaking down the total execution time in the different tasks inside the script we can check in which part to focus our optimization efforts. Figure 5.7 shows a break down of all the tasks and their execution time.

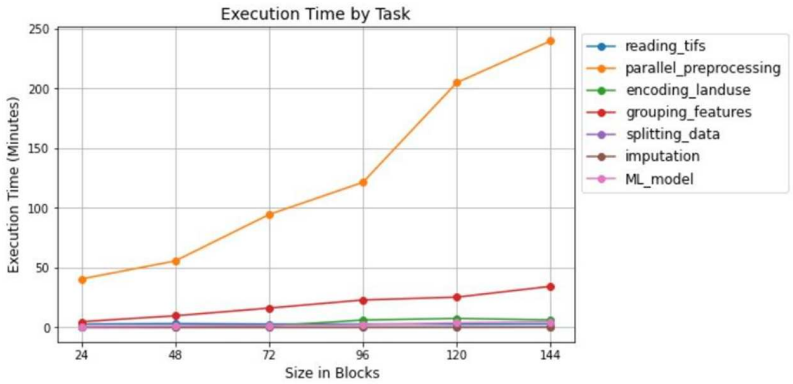


Figure 5.7: Script execution time breakdown for a different amount of blocks.

The process of preprocessing the blocks is the most costly one and also scales with the amount of blocks used, so we focused all the optimization efforts on it. Most of the other tasks on the other hand hardly scale with respect to the number of blocks used. Although grouping features slightly scales with respect to the number of processed blocks, we did not consider it significant enough to spend time optimizing that task.

Figure 5.8 shows that, as the block preprocessing's task scales with the number

of blocks while the other tasks take a relatively constant time, the percentage of the execution time dedicated to preprocessing the blocks will also increase until reaching almost 100%.

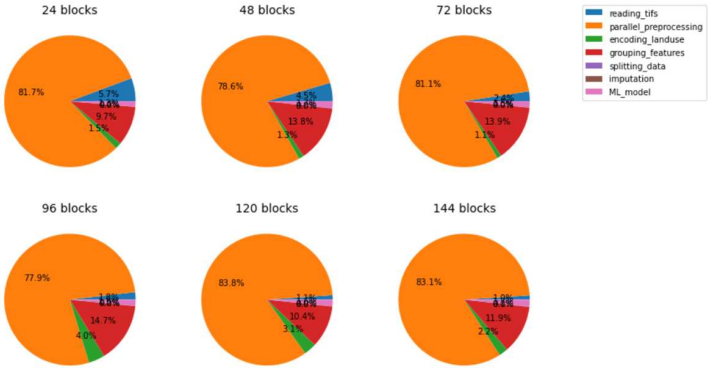


Figure 5.8: Piecharts showing the percentage of the script's execution time that each task takes.

However, when we tested how our model performed as the number of blocks increased, we found out that it was very inconsistent as shown in Figure 5.9.

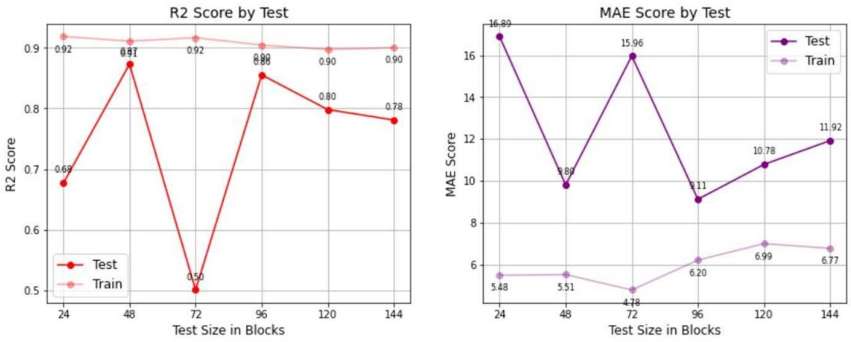


Figure 5.9: R^2 and MAE score comparison as the number of used blocks increases.

If we look only at the testing scores, we can see that the model is very inconsistent, fluctuating between good scores and bad scores regardless of the number of blocks used. On the other hand, the training scores are rather good and consistent, although they do not improve as the number of blocks used increases. Checking at the training and testing splits together, we can conclude that the

model is clearly overfitting to the training data and thus memorizing instead of learning from our data, resulting in poor performances on the testing partitions. We will work on solving that problem on next sections, mainly by reducing the number of features of the model and adding a regularization parameter to our model. The fact that even the training metrics do not increase according to the number of blocks used leads us to think that this subsample of Catalonia is not large enough to see an important improvement on the accuracy (note that still it is only a 1.7% of Catalonia).

We also performed another test to see if our problem was memory bound. The test involved processing blocks using the same number of blocks as CPUs in two different ways: first using the same block on each CPU, and then using different blocks on each CPU. When using the same block, the CPUs do not need to compete for resources as a block can fit on memory, but when using different blocks, if there is not enough memory CPUs will have to compete for memory and thus the execution time will increase. Note that in order to ensure a rigorous test, we had to find blocks of a similar size and thus used small sized blocks.

Figure 5.10 shows the results of the test. Until 6 CPUs, both methods perform the same, with the execution times being very similar. However, From 12 CPUs on, the execution time when processing different blocks is higher, which means that the CPUs do not have enough memory to fit all the blocks and thus the program is not going as fast as it could.

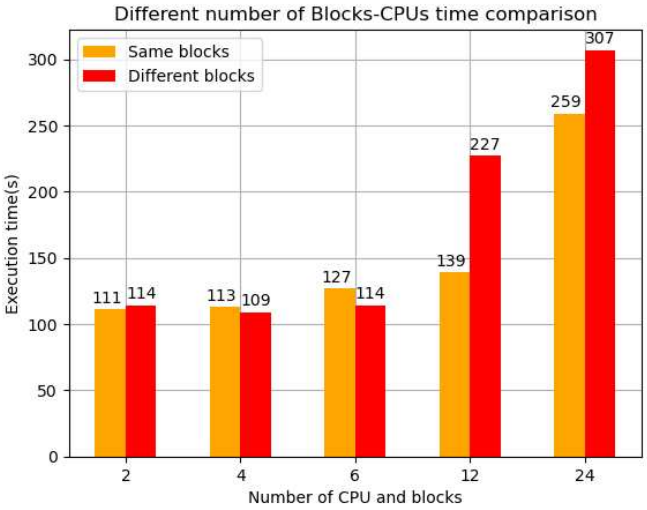


Figure 5.10: Bar chart showing the different execution times on cases where the same block is processed and where different blocks are processed.

When we optimized the memory usage, we managed to get a 5% reduction for each 24 blocks, which is not high enough but helped us slightly optimizing the execution time.

5.2.3 Feature Selection

The results of the process can be seen in Figure 5.11:

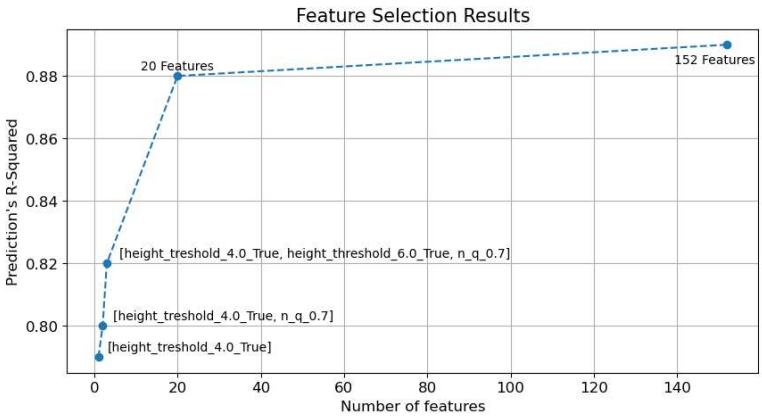


Figure 5.11: Results of Feature Selection on Canopy Cover prediction.

As the number of features decrease, the coefficient of determination for the testing set also decreases. For the first 132 rejected features there is only a R^2 decrease of 0.01, which we have considered to be acceptable taking into consideration that the model was almost 8 times simpler. For a lower number of features we saw that the model still performed reasonably good, specially with one feature, as it managed to reach a 0.79 R^2 score.

Note that the "height_threshold_x_True" feature keeps track of the relative amount of datapoints above x meters considering only the vegetation points. On the other hand, the feature "n_q_0.7" keeps track of what is the 70% percentile for the "number of returns" feature.

In order to keep the performance as high as possible while reducing the number of features, we decided to keep 20 features, which are:

`'height_q_0.2', 'height_q_0.3', height_threshold_3.0_True', 'height_threshold_4.0_True', 'height_threshold_5.0_True', 'height_threshold_6.0_True', 'height_threshold_7.0_True', 'n_q_0.3', 'n_q_0.4', 'n_q_0.5', 'n_q_0.6', 'n_q_0.7', 'n_sd', 'i_threshold_10_True', 'i_threshold_130_False', 'i_threshold_70_False', 'num_points_5', 'num_points_2', 'r_threshold_1_False', ‘slope'`

The first two features show the percentile 30 of the height feature, while the others are related with the relative amount of points above a certain threshold. For the number of returns feature, the model have decided to keep track of the 30, 40, 50, 60 and 70 percentiles as well as the standard deviation of it. Looking at the intensity, a high threshold value (130) and a small one (10) along with a medium value (70) have been kept. Also the amount of points that belong to class 2 and 5 seem to be important, as those classes are the ones corresponding to Ground and High Vegetation.
Related to the number of returns, we also have the return number, which is also used as a feature, measuring the amount that surpasses a threshold of 1. Finally, the slope seems to be also important, as a terrain with steeper slope might indicate mountains, which usually have higher Canopy Cover.

From the feature selection we can see that the most important feature has been the height, with a total of 7 features, followed by the number of returns

with a total of 6. Intensity seems to also be important, as 3 of the features are calculated from it. Finally, the percentage of "high vegetation" points and "ground" points also bring information to the model, along with slope and the return number.

In conclusion, the first model contained 152 features, which provided a high execution time, high memory usage, high redundancy, high overfitting and less explainability, while the feature selection allowed us to keep only 20 features (less than 15% of the initial features), providing low execution time, no redundancy, no overfitting and high explainability.
This step allowed us to process the whole Catalonia in a reasonable time, as processing 24 blocks before the Feature Selection took around 1 hour, which was reduced to 2 minutes (3.33% of the total time) after keeping only 20 features. This meant that in the time we processed 24 blocks before, now we were processing 30 times more, 720 blocks.

5.2.4 Hyperparameter Tuning

After spending many days performing hyperparameter tuning using Random Search with Cross Validation, we found out that the combination that showed a better performance was:

```
colsample_bytree=0.75
gamma=0
learning_rate=0.1
max_depth=25
min_child_weight=25
n_estimators=1000
reg_alpha = 100
reg_lambda = 2
subsample = 0.1
```

This hyperparameters chosen for the XGBoost model indicate a tailored and thoughtful approach to balancing model complexity, learning capacity, and regularization. With 1000 estimators, the model is given substantial opportunity to learn from the data, allowing for nuanced and detailed decision-making.

The max_depth of 25 suggests that each tree in the ensemble can be very deep, allowing the model to capture intricate patterns and interactions in the data. Such depth is beneficial in scenarios where the underlying relationships are complex and multi-faceted. However, deep trees can also lead to overfitting, capturing noise rather than signal. This necessitates robust regularization strategies to ensure generalization.

In this setup, the regularization is achieved through a combination of reg_alpha and reg_lambda. The reg_alpha parameter is set to a high value of 100, introducing substantial L1 regularization. This helps in shrinking the coefficients of less important features to zero, thereby simplifying the model and preventing overfitting. Meanwhile, reg_lambda, set to 2, applies L2 regularization, further stabilizing the learning process by penalizing large weights.

The learning_rate of 0.1 indicates a cautious and incremental approach to model updates, ensuring that each step towards the optimal solution is measured and less prone to drastic fluctuations. This slower learning rate helps in making the training process more robust and less likely to overshoot local minima.

Additionally, the model employs a min_child_weight of 25, which ensures that the nodes in the tree are not split unless they contain at least 25 instances. This prevents the model from learning from small, potentially noisy samples (such as low LiDAR density areas), thus contributing to its overall robustness.

The colsample_bytree parameter is set to 0.75, indicating that each tree is built using 75% of the features. This randomness in feature selection helps in reducing the correlation among trees, thereby enhancing the model's generalization ability.

Lastly, the subsample parameter is set to a low value of 0.1, meaning that each tree is trained on only 10% of the data. This aggressive subsampling significantly boosts regularization, further combating overfitting. Such a strategy can be particularly useful in high variance data scenarios, ensuring that the model does not become too tailored to the training data.

In summary, the chosen hyperparameters reflect a sophisticated balancing act between leveraging the powerful learning capabilities of XGBoost while employing strong regularization to maintain generalization. The high number of estimators, deep trees, and incremental learning rate, combined with significant regularization

and subsampling, make this setup well-suited for capturing complex patterns without succumbing to overfitting, which was mandatory for our canopy cover prediction.

5.2.5 Final model

The full model was trained after all the mentioned optimizations, feature selection and hyperparameter tuning. The processing time of all Catalonia (8424 blocks) was of 4 hour and 30 minutes, but only the 80% of it (6739) was used for training the model. This training took 46 minutes, and the final model weights around 200 Mb. Afterwards, we assessed the performance of the model using the remaining 20% of the blocks (1685).

The following table summarizes the metrics for the final model:

	R2	**MAE**
Training Set	0.89	6.30
Testing Set	0.88	6.47

Our model has achieved a high performance, reaching a 0.88 score on the coefficient of determination, and a mean absolute error of only 6.47, while keeping the training and testing metrics close, which indicates that the model was properly fitted and we managed to significantly reduce overfitting.

We chose 3 blocks from the Diverse10 dataset (that were not used in any of the training of the models) to visualize the predictions made by the different models we trained. Figures 7.4, 7.4 and 7.4 in the appendix show the real Canopy Cover compared with the prediction made with a model trained with 80% of Catalonia (Full model) along with the prediction made using a model trained with only 115 random blocks (Partial model), which represent approximately 1.3% of Catalonia.

Notably, the predictions generated by the full model exhibit a remarkable level of accuracy, closely aligning with the actual values.

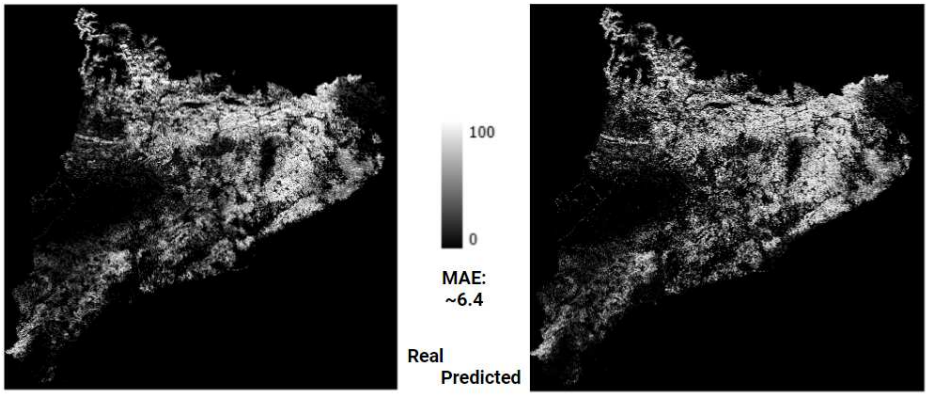
In contrast, the predictions derived from the partial model depict a noticeable divergence from the real values, indicating a comparatively inferior predictive capability. The higher Mean Absolute Error observed in the partial model predictions corroborates this observation, suggesting that the model's performance degrades when trained on a substantially reduced dataset.

While the shapes of the predicted values by the partial model may appear satisfactory, the discernible discrepancy in the actual numerical values further emphasizes the limitations introduced by the smaller training dataset. This discrepancy between predicted and actual values underscores the importance of data volume in training robust and accurate machine learning models.

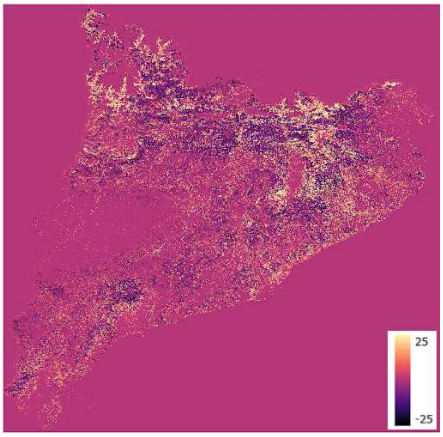
Figure 5.12 shows a broader view of our model's performance along Catalonia, as it shows the real and predicted Canopy Cover map for all Catalonia, along with the difference between the predicted value and the real value, showing the areas where the model underestimated and overestimated, which can give important insights about how the model treats different landscapes.

If we check the difference, we can see that the model has an excellent performance in the middle west of Catalonia, while in the north it is where it fails the most, probably due to the mountains and their irregular landscape. The coast has a small and consistent error, mostly overestimating the canopy cover value.

After having trained the final model, we used it along the built software to predict the whole Catalonia's Canopy Cover, as showed in Figure 5.12, with a processing time of 4.5 hours and a prediction time of less than 1 hour.



(a) Catalonia's real and predicted Canopy Cover



(b) Catalonia's Predicted-Real Canopy Cover

Figure 5.12: Catalonia's Canopy Cover comparison after predicting with the full model.

Chapter 6

Discussion

6.1 Error analysis

In order to get a broader understanding of the model's performance, we will be analyzing the errors that our model has been doing when predicting Canopy Cover in the whole dataset.

Figure 6.1 shows the error distribution (real values - predicted values) in a frequency histogram binned in groups of 2%.

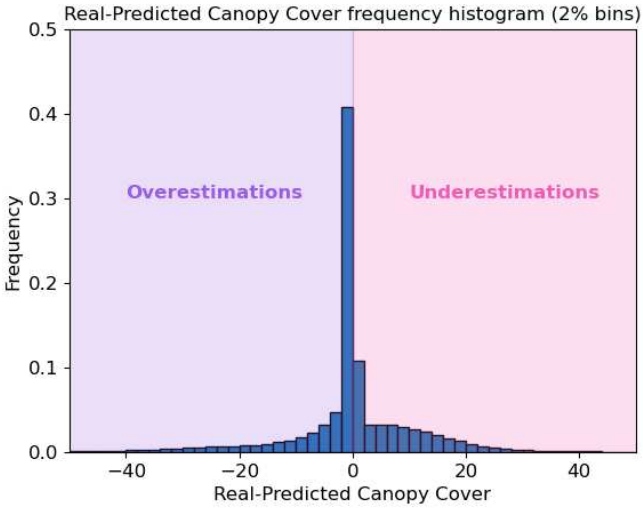


Figure 6.1: Error distribution histogram.

The errors seem to be close to normally distributed, with a high peak around the zero error. Taking a closer look, the graph shows that more than 50% of the predictions made have an error very close to zero, specifically into the (-2,2) error range. As the error moves farther from zero, the number of times the model did that error gets asymptotically smaller. In fact, the amount of times our model has made an error higher than |25| is very close to zero.

Figure 6.2 shows a confusion matrix where the predictions and real values have been grouped into bins of 10 units, showing how the model performs on different ranges of values.

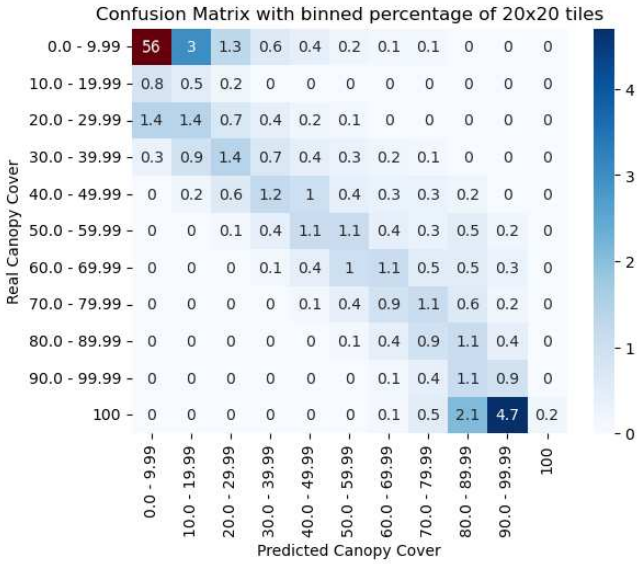


Figure 6.2: Discreted Prediction confusion matrix.

Note that the first cell has been artificially painted as its large scale made all other cells be of the same color.

Around 60% of Catalonia seems to have less than 10% Canopy Cover, most of it has been correctly predicted. Many of the other cells are close to zero, with the highest numbers of each row usually being on and around the diagonal. This fact indicates that our model is mostly predicting the correct value or at least near it, as the matrix is close to sparse.

Note as well that our model seems to have a tendency to underestimate. This behavior results in the confusion matrix having large values on the lower diagonal. We have the hypothesis that the provided groundtruth had been overestimated provided that we manually checked some of the 20×20 tiles and the real percentage of canopy cover was in many occasions lower than the described by the groundtruth. Then, our model making a slight underestimation might actually help reaching a closer to reality value than the actual groundtruth.

Some of the highest errors might be due to low density areas, as there are some areas that contain very low density such as the observed in the figure 3.9.

6.2 Extrapolating to higher densities

Our LiDAR data ensured a 0.5 points / m^2 density on 95% of the blocks, but more modern LiDAR censors are increasing this parameter to much higher densities. In fact, the LiDAR that will be used by firefighters have a density of 90 points/m^2, which is 180 times greater than the LiDAR used to train the model.

However, our model is able to make a canopy cover prediction of the blocks regardless of the point densities due to the fact that the aggregates are based on "which percentage of points meet a criteria" instead of "how many points meet a criteria". Moreover, with higher densities our model should be able to yield better predictions as the data is more accurate and there is more information.

The aim of this section is to measure how does the processing time increase as the LiDAR input data is denser.

Figure 6.3 shows how does the execution time increase for different tasks given the input size in MegaBytes.

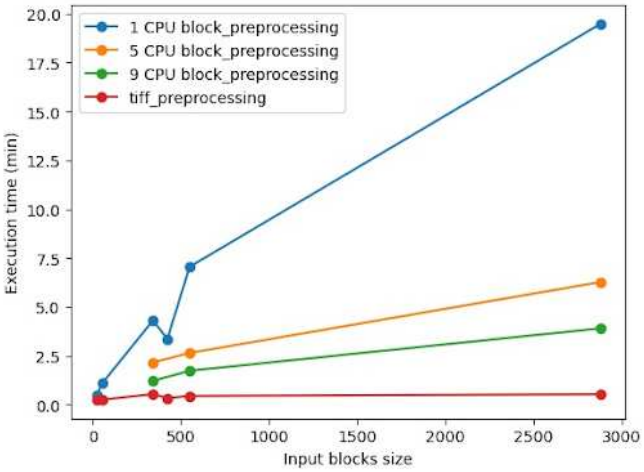


Figure 6.3: Execution time variation when increasing the block density.

The lower sizes correspond to a density of 0.5 points/m^2 (Catalonia), while sizes in the middle correspond to densities of 6 points/m^2 (Canada) and largest sizes correspond to densities of 10 points/m^2 (Navarra).

The graph shows that the execution time increases from less than 1 minute to almost 20 minutes when the block size increases. The high impact of executing the program in parallel is another important feature, as only by using 5 CPU, the execution time is lowered by three times, which remarks the parallelization capabilities of the model.

6.3 Problems faced during the project

Although I am very happy with the results of the project, we have encountered many problems during the development that required many hours to be solved and even some of them could not be completely worked out.

6.3.1 Zero Canopy Cover

As shown in figures such as 5.5 or 5.2, our models tended to misclassify many points that were labeled as 0% Canopy Cover, assigning them a suspiciously high

value. However, this effect was not seen the other way around (i.e. there were hardly no points classified as 0 that were not 0). This fact made us think that there was some kind of irregularity on the groundtruth map.

Figure 6.4 shows a satellite view of a specific Catalonian zone, along with the Canopy Cover map overlapped. Surprisingly, there are some 20×20m tiles that seem to contain tree canopy and thus the canopy cover should be higher than 0. However, the groundtruth has defined it as 0% Canopy Cover.

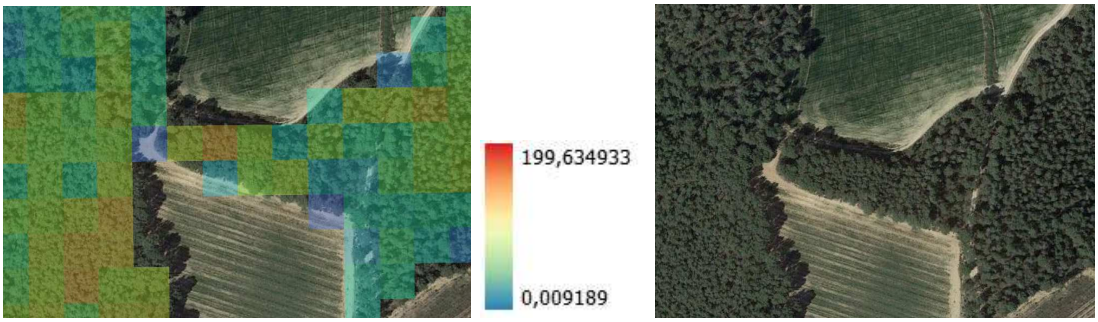


Figure 6.4: Satellite view of an area along the canopy cover assigned by the groundtruth. Note that in the first picture, all the tiles that do not have a color assigned have been labeled with 0% canopy cover.

For mitigating this effect we tried to build a model that would first classify a tile between 0-valued or non-zero valued, and then the non-zero valued tiles would go through the main model to assign them a value higher than zero. However, this approach did not work as good as expected, mainly because the errors were on the groundtruth and thus LiDAR did not contain any pattern for detecting the 0 values.

This errors then propagate to the model, as when the training labels are mistaken, the model is learning from wrong targets and thus will transmit those errors to the predictions.

In the future, the model could be improved with a more rigorous groundtruth.

Another possible solution would be to detect those mistaken zero valued tiles using outlier detection and leaving them out of the training process, which could be a future implementation.

6.3.2 Converting LiDAR data to tabular data

One of the main problems that we encountered in the beginning was that for each 20×20m tile we had an irregular number of points and thus the dataset had an inconsistent shape. For training a machine learning model, we needed a tabulated dataset where each row was assigned a target label. As we could not directly feed the model with the singular LiDAR points (inconsistent shape, too many columns, ...) we decided to design our dataframe such that we would use aggregates of the points inside each tile to gather up information that would give a solid representation of the area which the model could then use to make predictions.

6.3.3 Tendencies

As shown in Figure 5.5, when we trained a model with many blocks, we started seeing two tendencies on the scatter plots. The points that belonged to one tendency used to be overestimated, while the points that belonged to the other one used to be underestimated.

Figure 6.5 shows the two tendencies coloured in a scatter plot of the NextToBlock dataset.

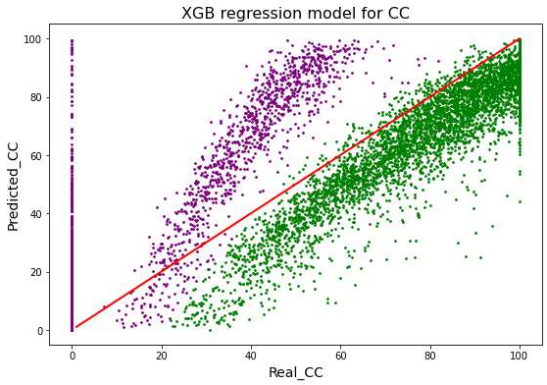


Figure 6.5: Scatter plot with the tendencies manually colored.

These two tendencies did not only show up on the results of the model, but also while comparing the the first component of the PCA with the Canopy Cover or when comparing a height attribute with the Canopy Cover as shown in Figure 6.6.

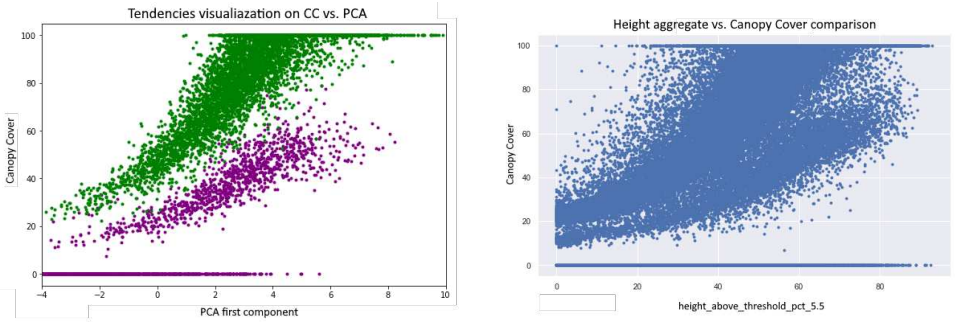


Figure 6.6: Visualization of the tendencies when comparing the first component of the PCA (left) and a height aggregate (right) with the Canopy Cover.

At first we thought that it might be only a random phenomena, but when we coloured each point from the block according to the tendency it belonged, we clearly saw that there was some kind of pattern as shown in Figure 6.7.

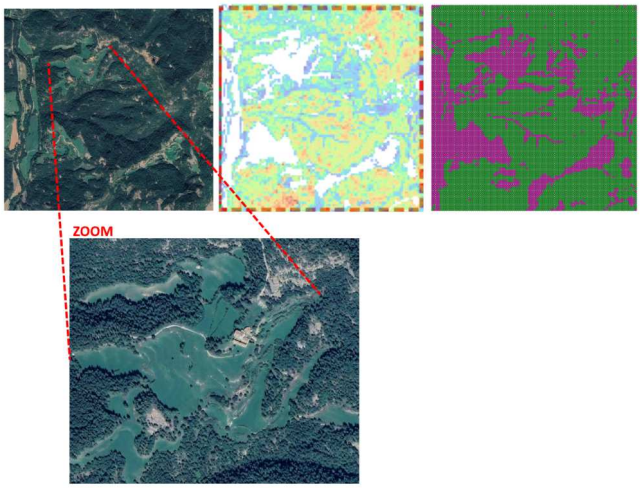


Figure 6.7: Google earth (left), Canopy Cover (middle) and tendencies (right) visualization of the NextToBlock dataset.

We figured out that the tendencies gathered around the points that had a low canopy cover value, but still there were some areas that contained a purple tendency that also had a high canopy cover value (between 20% and 60%). Figure 6.8 shows the same block, this time the canopy cover that had a value close to 0 was painted in blue.

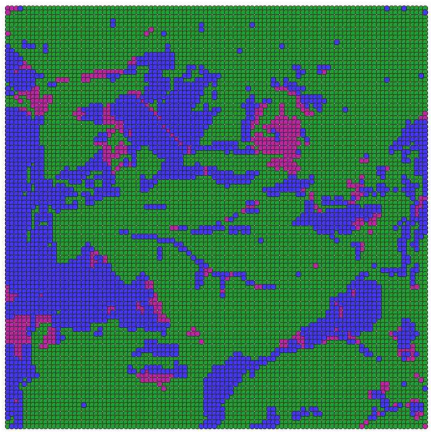


Figure 6.8: NextToBlock tendencies with the zero valued canopy cover painted in blue.

We can see that still some areas contain a high canopy cover and are labeled

as the purple tendency.

However, this tendencies issue did not only show up on that dataset, as all other datasets also had a similar problem. For instance, check the fullblock dataset when comparing the first component of the PCA with the Canopy Cover in Figure 6.9 and the map visualization of the Canopy Cover and the tendencies in Figure 6.10.

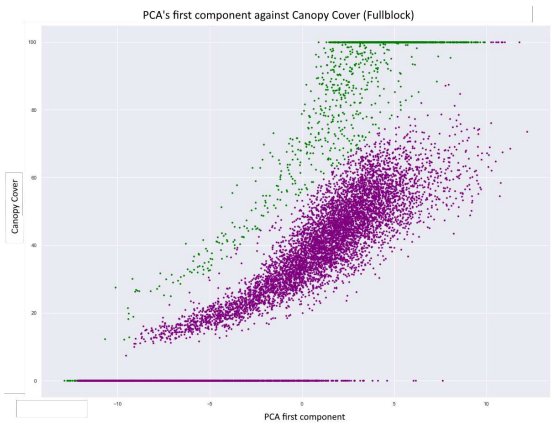


Figure 6.9: Tendencies visualized for the fullblock dataset when comparing the first component of the PCA with the Canopy Cover.

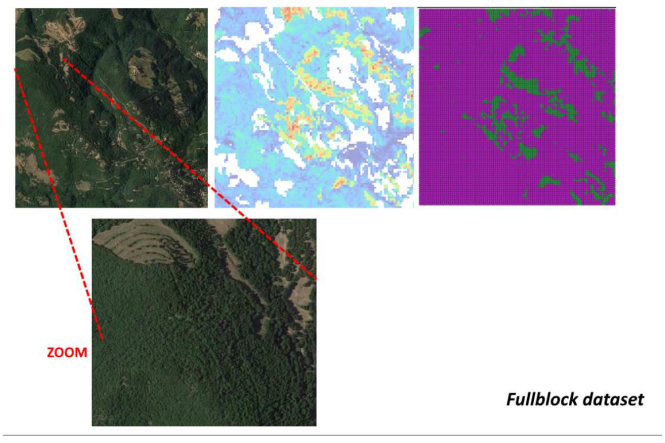


Figure 6.10: Google earth (left), Canopy Cover (middle) and tendencies (right) visualization of the Fullblock dataset.

We made some hypothesis of where could both tendencies come from, for example different types of vegetation that we cannot see (deciduous and evergreen), or the time when the LiDAR data was recorded (maybe some areas were recorded in Autumn and thus the trees did not have leafs then). However, we quickly discarted that hypothesis when we compared the different flying times that the NextToBlock datased had as shown in Figure 6.11.

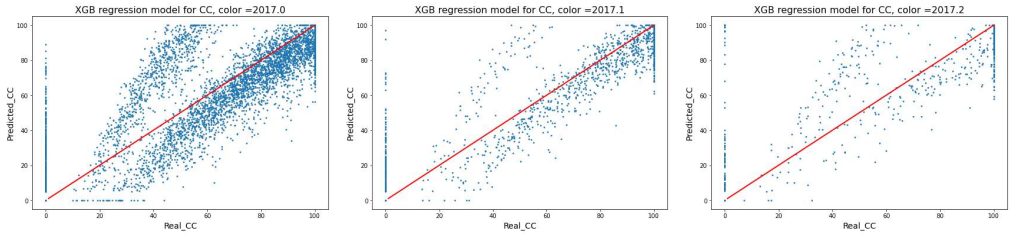


Figure 6.11: Scatter plot differentiating the points by the time they were measured.

As the tendencies showed in all three LiDAR measuring moments we rejected the hypothesis.

Finally, our most accepted hypothesis was that the model was wrongly learning from the incorrect zero valued Canopy Cover of the groundtruth and thus it had a confusion when predicting low valued canopy cover, leading up to having two tendencies show up, specially on lower values. In fact, for higher values of the canopy cover (>60%) the model shows a good performance and does not hesitate between multiple tendencies.

6.3.4 Memory and Execution Time issues

Another of the major problems that we encountered occurred when scaling the data, as even using the cluster we still had memory problems and a high execution time. We dedicated most of our efforts on solving those problems as shown in the "Cluster" section, but each time we wanted to test our optimizations we had to wait a significant amount of time due to the costly executions, making the whole process very tedious and slow.

6.3.5 Block overfitting

At one point we had to make a design decision on how to divide our data for training an testing. One option was to mix all the tiles from all the blocks and randomly splitting them into training and test. However, when we tried that option we saw that there was a large overfitting as the model was memorizing the 20×20m tiles and was not generalizing. As a solution, we chose to group the tiles in their original blocks and split those blocks into train and testing. Then, we would use all the tiles inside the training blocks to train the model and all the tiles into the testing block to evaluate the performance.

Chapter 7

Conclusions

7.1 Comparison with literature

While all reviewed studies focus on predicting Canopy Cover from LiDAR data, direct performance comparisons are challenging due to variations in geographical modeling difficulties and methodological differences. Nevertheless, their metric values provide useful benchmarks to assess whether our model's performance is within a normal range or significantly deviates.

Narine et al. achieved R^2 scores of 0.50, 0.61, 0.93, 0.75, 0.63, 0.72, 0.84, 0.77, 0.70, and 0.79 using ICESat-2 LiDAR data across multiple studies [30] [29] [28]. These scores highlight the variability in predictive accuracy depending on the specific geographic and environmental conditions of the study areas. Saleh et al. reported an R^2 score of 0.663 by integrating LiDAR data with Landsat 8 OLI imagery, demonstrating the potential benefits of combining different types of remote sensing data to enhance model performance [45]. Similarly, Tang et al. obtained R^2 scores of 0.27, 0.57, 0.45, and 0.30 by evaluating different models in various global regions [51]. These relatively lower scores may reflect the increased complexity and variability inherent in a broader, more diverse range of study sites. Posilero et al. achieved an R^2 score of 0.562 using linear regression techniques [38], indicating that simpler models can still yield reasonable predictions but may lack the precision of more sophisticated approaches.

Most models appear to achieve R^2 scores around 0.60 to 0.70 with LiDAR point densities of 5 points/m^2. In contrast, our model achieved an R^2 of 0.88 using point densities of 0.5 points/m^2. This significant improvement suggests a higher prediction power while utilizing fewer resources, highlighting the efficiency and effectiveness of our approach. Our model's performance indicates that it can produce highly accurate canopy cover predictions even with lower LiDAR data densities, which can be particularly advantageous in terms of cost and computational efficiency.

Overall, our findings suggest that while geographic and methodological differences must be considered, our model demonstrates a robust capability for canopy cover prediction, outperforming many existing models in the literature. This performance underscores the potential for further refining and optimizing remote sensing models to achieve even greater accuracy and resource efficiency.

7.2 Summary and conclusions

In conclusion, this research successfully accomplished its primary goals, significantly advancing the prediction of CC for Catalonia through the application of machine learning and artificial intelligence. The implementation of these technologies resulted in a mean absolute error (MAE) of 6.47, achieved in less than an hour, which is remarkably efficient compared to the longer time frames required by traditional methods employed by the ICGC. This efficiency underscores the practicality and effectiveness of our approach.

The project addressed the critical need for timely and accurate CC predictions, which are essential for predicting forest spread and managing forest resources. The developed model not only provided updated CC values but did so with a speed and precision that make it an valuable tool for forest management strategies.

One of the key objectives was to create a versatile framework capable of ex-

tending beyond CC to other biophysical variables. The model's successful use of LiDAR data to generate maps swiftly paves the way for broader applications in resource management and environmental monitoring. This adaptability marks a significant contribution to the fields of remote sensing and geospatial analysis.

The study also aimed to ensure that the solution was accessible and practical for various computing environments. With a software size of less than 300MB and parallelization capabilities, the developed framework is highly portable and efficient, making it feasible for diverse computational platforms from standard machines to resource-constrained systems.

In essence, our research not only addresses the specific challenge of Canopy Cover prediction for Catalonia but also lays the foundation for a versatile and efficient framework applicable to a spectrum of biophysical variables. The amalgamation of speed, accuracy, and accessibility positions our approach as a valuable asset in advancing the capabilities of geospatial analysis and environmental monitoring.

7.3 Further Work

One of the most important parameter of our model, the height, which takes up to 7 out of the 20 total features has been estimated in a very simple way. It would suppose a large improvement if we could manage to making a better approximation of the heights of our points, either by using outside data or applying complex DEM algorithms.

The performance achieved can be enhanced by adding extra information to our model. An approach would be to add ortophotos to our model's input, and changing the architecture to a multi-modal neural network composed of tabular data (MLP) and images (CNN), or even building a multi modal transformer.

Otherwise, LiDAR data can be converted into images by coloring the pixels by class, as done when visualizing the data in QGIS. This way the LiDAR data

could be used not only as tabular but also as image data, keeping the software with as few inputs as possible.

Furthermore, for the sake of following the main goal of the project, it would be interesting to build only a model based on LiDAR data (in our case we have also used slope data).

Moreover, on this article we have only presented a model that was able to predict Canopy Cover. Instead, many other biophysical variables can be predicted, such as fuel types, tree heights or leaf biomass, using a multiple output model that could calculate all the necessary biophysical variables that the FARSITE software requires.

7.4 Personal Reflections

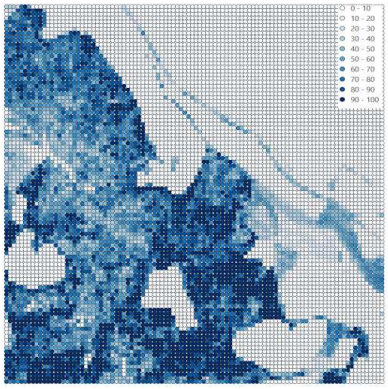
This has been a very rewarding experience. Firstly, I think that it has been a good contact within the research field, which has always emerged some interest on me. Also, I have had the chance to broaden my knowledge thanks to the problems that had to be solved and to apply the knowledge learned at college in a more practical way, as in the majority of cases at the university are worked in a more theoretical manner o making assumptions that move away the problem from reality. Working on the research environment requires you to get deeply informed about the methods that will be used to ensure that they do not have any restrictions for obtaining rigorous and valid results.

It has been a nice experience to share department with some of the best minds of the college, and I have learned a lot on the meetings, not only from the feedback but also on how to better expose the results, discussing the problems and presenting the updates.

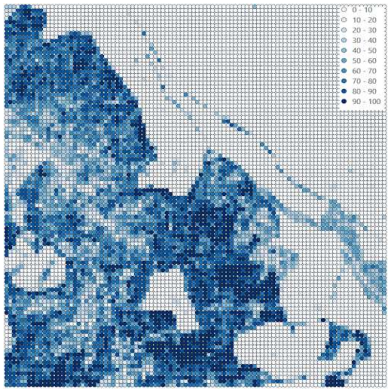
I am very satisfied on the outcomes of the project, not only on the model's performance but also on the acquired aptitudes.

Furthermore, I am happy to write a scientific article and leave my footprint on the research community. Moreover, I hope the work done can impact the resolution of fire spreads, hopefully reducing the damage done.

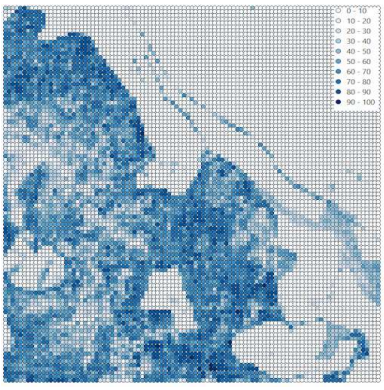
Appendix: Block prediction visualization



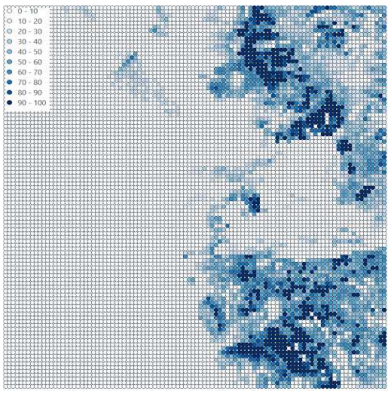
(a) Real CC.



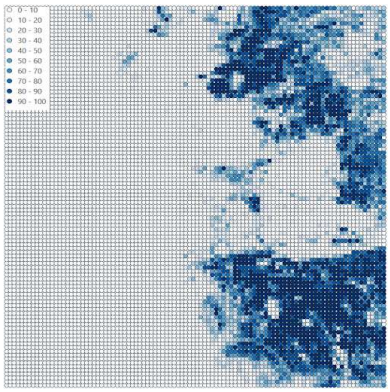
(b) Predicted with Full model (6 MAE).



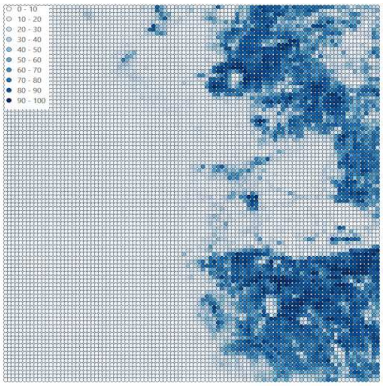
(c) Predicted with Partial model (15 MAE).



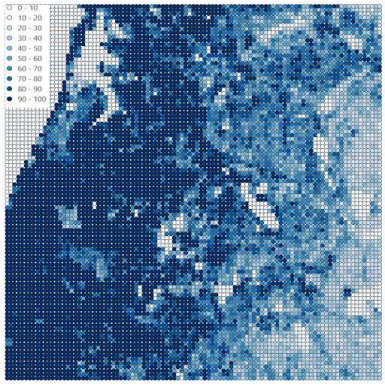
(d) Real CC.



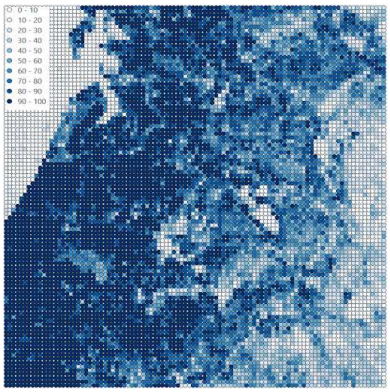
(e) Predicted with Full model (8.8 MAE).



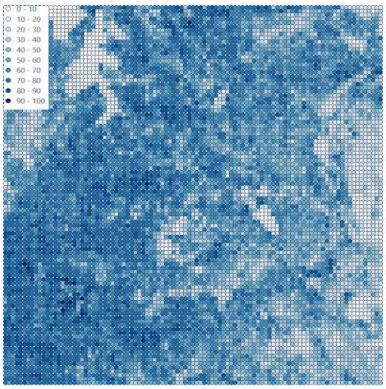
(f) Predicted with Partial model (12.5 MAE).



(g) Real CC.



(h) Predicted with Full model (7 MAE).



(i) Predicted with Partial model (10.5 MAE).

References

[1] H. E. Anderson. Aids to determining fuel models for estimating fire behavior. *US Department of Agriculture, Forest Service, General Technical Report*, 1982. 29

[2] 2022 Artificial Intelligence for Renewable Energy Systems. Mean absolute error. https://www.sciencedirect.com/topics/engineering/mean-absolute-error, 2024. 51, 52

[3] Jason Brownlee. Why one-hot encode data in machine learning? https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/, 2020. 56

[4] Qi Chen. Airborne lidar data processing and information extraction, 2007. 14

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 49

[6] Kathleen Coupland, David Hamilton, and Verena C. Griess. Combining aerial photos and lidar data to detect canopy cover change in urban forests. *PLoS ONE*, 17, 9 2022. 10

[7] dask. Parallel python fast and easy. https://www.dask.org/, 2024. 61

[8] DMLC. XGBoost python package documentation. https://xgboost.readthedocs.io/en/stable/python/index.html, 2022. Accessed: July 17, 2023. 50, 57

[9] Python docs. multiprocessing — process-based parallelism. https://docs.python.org/3/library/multiprocessing.html, 2024. 62

[10] ESRI. Sentinel-2 10m land use/land cover time series of the world. produced by impact observatory and esri. https://www.arcgis.com/home/item.html?id=cfcb7609de5f478eb7666240902d4d3d, 2023. 27

[11] Goran Krsnik et al. Regional level data server for fire hazard evaluation and fuel treatments planning. *Remote Sensing*, 2020. 28, 29

[12] freeCodeCamp Ihechikara Vincent Abba. What is r squared? r2 value meaning and definition. https://www.freecodecamp.org/news/what-is-r-squared-r2-value-meaning-and-definition/#:~:text=What%20Does%20an%20R%20Squared,the%20dependent%20and%20independent%20variables., 2023. 51

[13] Elizabeth A Freeman, Gretchen G Moisen, John W Coulston, and Barry T Wilson. Random forests and stochastic gradient boosting for predicting tree canopy cover: comparing tuning processes and model performance. *Canadian Journal of Forest Research*, 46(3):323–339, 2016. 2

[14] R. Gaulton and T. J. Malthus. Lidar mapping of canopy gaps in continuous cover forests: A comparison of canopy height model and point cloud based techniques. *International Journal of Remote Sensing*, 31, 2010. 7

[15] GDAL. Gdal. https://gdal.org/index.html, 2024. 33

[16] geeksforgeeks. Cross validation in machine learning. https://www.geeksforgeeks.org/cross-validation-machine-learning/, 2023. 58

[17] geeksforgeeks. Data normalization machine learning. https://www.geeksforgeeks.org/what-is-data-normalization/, 2023. 52

[18] geeksforgeeks. Hyperparameter tuning. https://www.geeksforgeeks.org/hyperparameter-tuning/, 2023. 56

[19] GISAT. Actionable intelligence from earth observation. https://www.gisat.cz/, 2023. 30

[20] google. Normalization. https://developers.google.com/machine-learning/data-prep/transform/normalization, 2024. 52

[21] HeavyAI. Feature selection. https://www.heavy.ai/technical-glossary/feature-selection#:~:text=Feature%20selection%20is%20the%20process,of%20datasets%20continue%20to%20grow., 2024. 53

[22] Institut Cartogràfic i Geològic de Catalunya. Dades lidar, models digitals d'elevacions i mapes de pendents. https://www.icgc.cat/Descarregues/Elevacions, 2020. Accessed: 2023-07-21. 28, 29

[23] Institut Cartogràfic i Geològic de Catalunya. Datos lidar - especificaciones. https://www.icgc.cat/es/Descargas/Elevaciones/Datos-lidar, 2023. 18, 24, 40

[24] Institut Cartogràfic i Geològic de Catalunya. Variables biofísiques de l'arbrat v1.1, 2023. 1, 13, 23

[25] ICGC. Modelo de elevaciones del terreno de cataluña 5 x 5 metros. https://catalegs.ide.cat/geonetwork/srv/api/records/model-digital-terreny-5x5-v1r0, 2018. 40

[26] Mapbox. Rasterio: access to geospatial raster data. https://rasterio.readthedocs.io/en/stable/, 2018. 55

[27] Shirley Moore. Amdahl's law, speedup, and efficiency. http://svmoore.pbworks.com/w/file/fetch/88888418/Speedup.pdf, 2014. 76

[28] Lana Narine, Lonesome Malambo, and Sorin Popescu. Characterizing canopy cover with icesat-2: A case study of southern forests in texas and alabama, usa. *Remote Sensing of Environment*, 281, 11 2022. 12, 97

[29] Lana L. Narine, Sorin Popescu, Amy Neuenschwander, Tan Zhou, Shruthi Srinivasan, and Kaitlin Harbeck. Estimating aboveground biomass and forest canopy cover with simulated icesat-2 data. *Remote Sensing of Environment*, 224:1–11, 4 2019. 11, 97

[30] Lana L. Narine, Sorin C. Popescu, and Lonesome Malambo. A methodological framework for mapping canopy cover using icesat-2 in the southern usa. *Remote Sensing*, 15, 3 2023. 7, 97

[31] Numpy. Numpy. https://en.wikipedia.org/wiki/NumPy, 2024. 65

[32] National Oceanic and Atmospheric Administration. What is lidar? https://oceanservice.noaa.gov/facts/lidar.html, 2023. Accessed: 2024-03-07. 17

[33] Padu. parallel-pandas 0.6.3. https://pypi.org/project/parallel-pandas/, 2024. 61

[34] Ebadat G. Parmehr, Marco Amati, and Clive S. Fraser. Mapping urban tree canopy cover using fused airborne lidar and satellite imagery data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-7:181–186, 6 2016. 8, 9

[35] Harshil Patel. Feature engineering explained. https://builtin.com/articles/feature-engineering#:~:text=Apr%2029%2C%202024-,Feature%20engineering%20is%20the%20process%20of%20selecting%2C%20manipulating%20and%20transforming,used%20in%20a%20predictive%20model., 2024. 39

[36] Joseph St Peter, Jason Drake, Paul Medley, and Victor Ibeanusi. Forest structural estimates derived using a practical, open-source lidar-processing workflow. *Remote Sensing*, 13, 12 2021. 7

[37] Andrew Plowright. Canopy analysis in r using forest tools. http://cran.nexr.com/web/packages/ForestTools/vignettes/treetopAnalysis.html, 2024. 14

[38] M A V Posilero, E C Paringit, R J L Argamosa, R A G Faelga, C A G Ibanez, and G P Zaragosa. Lidar-based canopy cover estimation using linear regression techniques. *Philippine Geoscience and Remote Sensing Society*, 2:26–33, 2016. 7, 11, 12, 97

[39] Previncat. Zones homogènies de regim d'incendis forestals. https://previncat.ctfc.cat/#download, 2024. 31

[40] Samuel "Jake" Price and Matthew J. Germino. Modeling of fire spread in sagebrush steppe using farsite: an approach to improving input data and simulation accuracy. *Fire Ecology*, 2022. 2

[41] QGIS. Qgis. a free and open source geographic information system. https://www.qgis.org/en/site/, 2024. 33

[42] rapidlasso. Lastools. https://lastools.github.io/, 2023. 34

[43] rapidlasso. Lastools. https://rapidlasso.de/product-overview/, 2023. 34

[44] LiDAR La Rioja. Calculation of the digital elevation model (dem). https://iderioja.github.io/doc_lidar_2016_processing_en/dtm_dsm/digital_elevation_model/, 2024. 40

[45] Muhammad Buce Saleh, Rosima Wati Dewi, Lilik Budi Prasetyo, and Nitya Ade Santi. Canopy cover estimation in lowland forest in south sumatera, using lidar and landsat 8 oli imagery. *Jurnal Manajemen Hutan Tropika*, 27:50–58, 4 2021. 7, 10, 97

[46] sammigachuhi. A brief exploration of lidar processing in python. https://medium.com/@sammigachuhi/a-brief-exploration-of-lidar-processing-in-python-304ba31287c3, 2024. 40

[47] Joe H. Scott and Robert E. Burgan. Standard fire behavior fuel models: A comprehensive set for use with rothermel's surface fire spread model. *USDA Forest Service - General Technical Report RMRS-GTR*, 2005. 29

[48] Simplilearn. Introduction to data imputation. https://www.simplilearn.com/data-imputation-article#:~:text=about%20Multiple%20Imputations.-,What%20Is%20Data%20Imputation%3F,from%20a%20dataset%20each%20time., 2023. 52

[49] sklearn. Simpleimputer. https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html, 2024. 52

[50] sklearn. Standardscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html, 2024. 52

[51] Hao Tang, John Armston, Steven Hancock, Suzanne Marselis, Scott Goetz, and Ralph Dubayah. Characterizing global forest canopy cover distribution using spaceborne lidar. *Remote Sensing of Environment*, 231, 9 2019. 9, 97

[52] tmontaigu. laspy: Python library for lidar las/laz io. https://laspy.readthedocs.io/en/latest/, 2018-2022. 53

[53] USGS. What is a digital elevation model (dem)? https://www.usgs.gov/faqs/what-digital-elevation-model-dem#:~:text=A%20Digital%20Elevation%20Model%20(DEM)

%20is%20a%20representation%20of%20the,from%20a%20variety%20of%20sources., 2024. 7, 40

[54] Shiva Verma. How fast numpy really is and why? https://towardsdatascience.com/how-fast-numpy-really-is-e9111df44347#:~:text=Because%20the%20Numpy%20array%20is,leap%20in%20terms%20of%20speed., 2019. 65

[55] Jochem Verrelst, Zbyněk Malenovský, Christiaan Van der Tol, Gustau Camps-Valls, Jean Philippe Gastellu-Etchegorry, Philip Lewis, Peter North, and Jose Moreno. Quantifying vegetation biophysical variables from imaging spectroscopy data: A review on retrieval methods, 2019. 1

[56] western fire chiefs association. How fast do wildfires spread? https://wfca.com/wildfire-articles/how-fast-do-wildfires-spread/#:~:text=Wildfires%20spread%20at%20an%20average%20of%2014.27%20miles%20per%20hour.&text=However%2C%20this%20can%20vary%20hugely,%2C%20fuel%20type%2C%20and%20terrain., 2024. 3

[57] Wikipedia. Dask (software). https://en.wikipedia.org/wiki/Dask_(software), 2024. 61

[58] Wikipedia. Google earth. https://en.wikipedia.org/wiki/Google_Earth, 2024. 34

[59] Wikipedia. Norm (mathematics). https://en.wikipedia.org/wiki/Norm_(mathematics), 2024. 64

[60] Wikipedia. Numpy. https://numpy.org/, 2024. 65

[61] Wikipedia. Speedup. https://en.wikipedia.org/wiki/Speedup, 2024. 76

[62] Wikipedia. Triangle inequality. https://en.wikipedia.org/wiki/Triangle_inequality, 2024. 64