POLITECNICO DI MILANO

Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in Computer Science and Engineering

Inferring High Resolution Terrain, Vegetation and Visibility Networks from Point Cloud Data

Relatore:
Prof. Pier Luca Lanzi
Correlatore:
Prof. Tanya Berger-Wolf

Tesi di laurea di:
Benedetto Vitale
Matr. 837125

Anno Accademico 2015 - 2016

Alla mia cara famiglia.

Ringraziamenti

Ringrazio profondamente il mio relatore Italiano Prof. Pier Luca Lanzi del Politecnico di Milano e la mia relatrice Americana Prof. Tanya Berger-Wolf dell'University of Illinois at Chicago. Il loro essere sempre disponibili quando necessario mi ha consentito di seguire il percorso che mi ha condotto al completamento dei miei studi.

Ringrazio anche Vena Jia Li, dottoranda dell'University of Illinois at Chicago, per esserci stata qualora avessi necessità di discutere eventuali problemi ed analizzare varie soluzioni agli stessi.

Un profondo ringraziamento va alla mia ragazza, Rosa, per avermi sempre supportato nonostante l'insormontabile distanza e sempre con un grande cuore.

Ringrazio sentitamente anche il mio amico d'infanzia Walter, per esserci sempre stato per me qualora necessario sin da quando eravamo bambini.

Infine ringrazio i miei amici e colleghi Italiani Andrea P., Andrea D.V., Davide, Ettore, Roberto and Vittorio, che hanno condiviso con me questo magnifico pecorso, aiutandomi a fronteggiare grandi quantità di lavoro con momenti di indimenticabile divertimento.

BV

List of Figures

List of Tables

Contents

Astratto

Con la diffusione di tecnologie che consentono l'acquisizione della struttura tridimensionale di varie superfici, è nato un cospicuo interesse per le informazioni apportate da tali tecnologie e per gli usi che sono possibili con questi dati. Un campo in cui questa tecnologia è stata ampiamente utilizzata è quello ecologico, dato che i dati 3D offrono la possibilità di analizzare la vegetazione e le sue caratteristiche direttamente dalla nuvola di punti, mediante tecniche mirate ad isolare i punti del terreno da quelli della vegetazione stessa, procedure finalizzate alla segmentazione di singoli alberi nella nuvola di punti ed analisi che utilizzano la struttura a punti derivata per stimare proprietà degli alberi stessi.

In questa tesi abbiamo proposto un framework che consente di estrarre informazioni sulla vegetazione e sul terreno a partire dalla nuvola di punti di varie aree naturali, classificandone prima i punti appartenenti al terreno o alla vegetazione tramite un algoritmo di curvatura proposto nella letteratura, poi individuando i punti appartenenti a singoli alberi utilizzando un algoritmo di segmentazione basato sulle distanze tra i punti stessi. Facendo leva sulle informazioni estratte per quanto riguarda terreno e vegetazione, il framework è in grado di stimare in due fasi l'ostruzione visiva degli elementi individuati tra la vegetazione, utilizzando *l'apprendimento supervisionato*. Durante la prima fase vengono estratte delle *caratteristiche* dai punti degli alberi individuati, utilizzando sia un approccio manuale, ovvero un gruppo specifico di funzioni viene usato per estrarre le caratteristiche, sia un approccio automatico tramite reti neurali in modo *non supervisionato*. Durante la seconda fase invece, diversi algoritmi di apprendimento supervisionato sono stati usati per stimare l'ostruzione visiva di ogni elemento della vegetazione utilizzando le caratteristiche estratte nella fase precedente. Abbiamo inoltre fornito un esempio pratico tra tutti i possibili usi delle informazioni estratte dal framework proposto, il quale consiste nell'utilizzare le informazioni estratte per costruire una rete di visibilità tra gli individui collocati nell'ambiente. Un altro possibile utilizzo delle informazioni estratte dal framework potrebbe essere fornire supporto nella creazione di un'esperienza *immersive* in ambienti forestali, esplorando l'habitat esattamente come gli animali stessi che vivono in esso. Il motivo dietro la generazione di una tale esperienza è che le interazioni tra gli animali della stessa o diversa specie influenzano significativamente il loro comportamento in molteplici aspetti, risultando pertanto cruciale negli studi del comportamento degli stessi.

Summary

With the diffusion of technologies which enables the acquisition of the 3-Dimensional (3D) structure of different surfaces, there has been a great focus on the information brought and the uses that are possible with this kind of data. One of the field in which this technology has been widely used is ecology, given that 3D data offers the possibility of analyzing trees and their characteristics directly from the point cloud structure by means of techniques aimed at isolating ground points from non-ground ones, procedures with the goal of segmenting individual trees in the cloud and analysis with respect to the trees point structure obtained in order to derive the trees properties.

In this thesis we proposed a framework which infers information on vegetation and terrain starting from raw point cloud data of forested environments, classifying first the points in the point cloud as ground and non-ground by means of a multiscale curvature algorithm proposed in the literature, then segmenting individual trees among the vegetation points by means of an innovative segmentation algorithm proposed. By leveraging on the information extracted for terrain and vegetation, the framework infers also the visual obstruction potential of the identified vegetation elements by means of a supervised inference approach. The first phase of the supervised inference consists in extracting a suitable set of features from the vegetation points structures, in which we tried to extract features both through a manual approach, meaning that we chose a specific set of features to extract, and through neural networks in an automatic way. In the second phase instead, different machine learning algorithms were used in order to infer the visual obstruction potential of each vegetation element using the features extracted in the previous phase. We also provided a practical example among the possible uses of the information extracted by the framework, which consists in exploiting the framework results to build a visibility network relative to the individuals located in the environment.

Another possible use of the information extracted by the framework could be providing support in generating an immersive experience in forested environments, experiencing the habitat exactly as animals living in it. The reason behind the generation of an immersive environment is that interactions among animals of the same or different specie significantly influence their behavior under many valuable aspects, thus being crucial in animal behavior studies.

Chapter 1

Introduction

Thanks to the availability of different technologies which enabled the extraction of the 3D point cloud structure from captured surfaces, there has been a great focus on what can be accomplished with 3D data in various fields. For example, point cloud data of forested surfaces has been used to estimate fruit-tree leaf areas [46] or to generate a vine plantation map [31]. Two are the main technologies used to obtain the 3D point cloud information relative to a certain scanned surface:

- **Laser Imaging Detection and Ranging (LIDAR)** is "an active remote sensing technology that measures properties of reflected light to determine range to a distant object" [17, 27]. "The range to an object is computed by measuring the time delay between the transmission of a laser pulse and the detection of the reflected signal" [17, 51]. Thanks to the possibility of generating a 3D structure with high resolution and accuracy, LIDAR has been widely used in fields such as ecology [46], geomorphology [31, 39] and remote sensing [28].
- **Aerial Photography** consists in using flying technologies such as drones to capture images of surfaces from different angles and at different heights. Thanks to the fact that each image covers a different height or angle of the surfaces, they can be assembled in order to obtain a detailed 3D point cloud structure of the captured surfaces, as successfully done in [16].

There are many advantages in analyzing the 3D structure of a certain scanned surface, which could be object identifications according to spatial features [45] or extracting charcteristics of vegetation elements in forested environments once obtained their points structure [19]. Another useful information which could be extracted from 3D point cloud data corresponding to forested environment derive from the analysis of the lines of sight among individuals located in the environment and of the interactions among them.

Visibility information is a crucial aspect in forested environments, especially in fields such as animal behavior, given the fact that interactions among animals of the same or different specie significantly influence their general behavior. Given the importance of visibility, using this information to generate an immersive environment

or to build a visibility network among animals would offer a great support to the community of scientists focused on animal behavior. It is quite hard to track animals in some natural environments due to their continuous movements and the fact that they often reach places where they are not easily observable, as with monkeys on very high trees. Having the possibility of experiencing forested environments exactly as animals do during their movements and everyday life would help significantly in understanding which are the causes that make animals behave or react in a specific way in their natural habitat. The objective of this thesis raises many technical challenges however, considering that plenty of information must be extracted from raw point cloud data.

1.1 Thesis Objectives

In this thesis we used the 3D point cloud data we implemented a framework extracting various information starting from raw point cloud data. We first infer information on the terrain by determining which points belong to the ground elements and which to the non-ground ones. Then we infer information about vegetation by detecting individual vegetation elements among the non-ground points identified. The proposed framework can be used for any ecological study, including those that focus solely on the vegetation (such as following the blooming and growth of trees over time). We showcase the use of this habitat information by applying it to compute the visibility network corresponding to the perspective of animals placed in that landscape. If animal position information is available, we can compute mutual visibility of animals in the habitat, given that their main obstructing features are on the scale of the features we can extract from the point cloud data.

At a high level, there are four main steps that must be performed in order to extract the information necessary to reach the final objective of this thesis:

1. **Ground/non-ground separation**: the point cloud data must be analyzed in order to identify points corresponding to the ground and points corresponding to non ground elements, such as vegetation. This step is crucial, given that detecting the right vegetation is fundamental for the functioning of the whole framework and for the correctness of the produced results.

2. **Tree segmentation**: points corresponding to vegetation must be segmented in individual vegetation elements. In doing so, it must be considered that a forest can present a huge variety of vegetation elements, thus it is necessary to perform a parameter space exploration to determine parameter settings that capture desired classes of vegetation most accurately.

3. **Visibility/obstruction characterization of vegetation**: points corresponding to vegetation elements must be analyzed in order to extract a suitable set of characteristics which would allow to infer the visual obstruction potential

of the vegetation element to which they correspond. Once those features are extracted, various machine learning classification techniques can be applied in order to infer the visual obstruction potential.

4. **Visibility network computation**: Given the location of two individuals, we must check whether some vegetation elements able to obstruct their lines of sight are located between those individuals. We use the results of the obstruction characterization of vegetation from previous step to infer whether two individuals can see each other and then to build a visibility network embedding the visibility information derived.

For some of these technical challenges, different approaches had already been proposed in the literature and we will have a deeper look at them in Chapter 2, highlighting advantages and limitations of the existing techniques.

For what concerns the structure of the thesis, we now describe the focus of each chapter:
Chapter 2 provides a review of the most recent solutions proposed in the areas of interest covered by our framework. Chapter 3 covers the background topics necessary to understand the work performed in this thesis. In Chapter 4 we describe the data used to perform the experiments in this thesis, their acquisition procedure and the information they provide. Chapter 5 shows in details the first step performed starting from the raw point cloud data, which is the ground/non-ground classification of points in the point cloud. In Chapter 6 we explain in a detailed way the segmentation technique used in this thesis, the underlying assumptions and the results obtained. Chapter 7 covers the features extraction procedures used and the experiments performed in order to infer the visual obstruction potential of the vegetation elements in the environment. In Chapter 8 we showcase a possible use of the information inferred with the proposed framework by deriving the dynamic visibility network among animals located in the environment at each timestamp. In Chapter 9 we make some considerations and reasoning on the presented methods, showing also potential applications and variants of the methods together with potential future developments.

Chapter 2

State of the Art

In this chapter we survey the existing work in the literature relevant to the objectives of habitat feature extraction from point cloud data and visibility calculations. Before proceeding, given that in our case the vegetation elements are characterized by trees, we will use the term "tree" instead of vegetation element when referring to vegetation.

2.1 Ground/Non-Ground Separation Approaches

When analyzing point cloud data, knowing which points correspond to ground elements and which to non-ground elements is fundamental in order to properly process data with the aim of extracting meaningful information. Given the importance of this operation, in the literature it is possible to find many different approaches aimed at properly performing this separation [7, 36, 12]. In [12] in particular the authors performed the ground/non-ground separation by using active shape models for the ground, which then were matched to the point cloud surface. However, estimating proper shapes for the ground is time consuming and requires prior knowledge of the environment. Given the impossibility of estimating shapes due to lack of information, for this thesis we used the *multiscale curvature algorithm for Point Cloud Data* proposed in [14], which is described in details in Chapter 5 and does not require any prior knowledge about the environment.

2.2 Tree Segmentation Approaches

Before the availability of laser scanning technologies, a typical method to extract 3D information of forested environments and capturing individual trees characteristics was field inventory, as done in [34]. However, field inventory can be really time consuming and limited by spatial accessibility, given that not all study areas can be accessed so easily. With the development of the 3D point cloud extraction technologies, various approaches making use of the 3D structure have been proposed, which

can be mostly grouped into two categories: the Canopy Height Model (CHM) based algorithms and the raw point cloud based ones.

2.2.1 CHM Based Algorithms

CHM-based approaches have in common the use of the point cloud derived CHM, which is a "raster image interpolated from the point cloud depicting the top of the vegetation canopy" [17]. While the CHM model brings a valuable amount of additional information, it may be affected by inherent errors caused by a various number of sources. For instance, it was shown that "a spatial error can be introduced during the interpolation process from the point cloud to the gridded height model, which can decrease the accuracy of tree segmentations and of the relevant measurements" [17]. Based on the CHM, various methods were proposed:
In [48] the authors proposed a tree segmentation algorithm with two main steps. First they adopted a local maxima detection approach to find the tree tops. Then, they used a region growing procedure to highlight the respective tree crowns, obtaining at the end the full trees. In [22] instead, the authors developed a multi-scale template matching approach in order to find individual trees in the environment. More in details, they adopted elliptical and other shaped templates in order to identify the points corresponding to a specific tree. However, this requires a priori knowledge of the tree shapes characterizing the forest at hand.

2.2.2 Raw Point Cloud Based Algorithms

In order to overcome the inherent errors related to the previous kind of approaches, algorithms operating directly on the raw point cloud data were proposed, making use only of the raw information provided by the point cloud elements. In the literature many approaches can be found:

In [38] the authors applied a clustering algorithm directly to the raw coordinates triple of the point cloud data (the raw coordinate of the three axis x, y, z). The clustering algorithm chosen was the *K-means* [33]. However, using the *K-means* implies that the results obtained will always be subject to the initialization points chosen for the clusters, which were estimated from the local maxima of a rasterized digital surface model, thus making use of more than just the raw point cloud data. In addition, it must be known which is the number of trees in the forest in order to properly choose the k parameter. In [24] instead, the authors developed a method based on two main steps. During the region growing step they identified subparts of the trees canopies, which then were merged into a full tree canopy by means of an agglomerative clustering approach. The most challenging part in such approach is the choice of the search radius R used to identify the treetops, which must be estimated by means of a trial and error approach. Another example of raw approach is [26], where the authors used an algorithm which identifies one tree at a time. In this method, they first found the point with the highest height in order to consider

it as a tree top, then they proceeded with decreasing height to identify the tree points by means of a minimum spacing rule and a spacing threshold. Once a tree is found, its points are removed by the initial point cloud. However, running such a method for a huge amount of points (typical of point cloud data) would be really time consuming, given that it is necessary to scan all the remaining points in the cloud in order to identify each tree.

In this thesis we used an approach similar to the one presented in [26] to perform the tree segmentation, but with different adopted criteria and thresholds in order to adapt the procedure to our setting.

2.3 Visual Obstruction Inference

After having identified the individual trees in the point cloud, a considerable amount of information could be extracted by analyzing the points structure of the trees. In the literature it is possible to find only more general approaches aimed either at identifying objects in the point cloud or inferring trees properties:
In [45] the authors tried to classify objects in the point cloud into roads, roof, grasses and trees by means of supervised parametric classification techniques, meaning that they already knew which was the set of object types in the point cloud. Another example is [19], where the authors tried to identify the tree species in the forest relative to the point cloud using the raw points and a rasterized image of the surface, where all the species present in the forest were already known. To the best of our knowledge, there are no studies in which the authors tried to infer visual obstruction potential or other aspects relative to visibility from trees.

2.4 Visibility Network Computation

Given that among the possible uses of the extracted information we chose to compute the visibility network relative to the individuals in the environment, we searched the existing literature for approaches making use of visibility networks in any setting. In the literature it is possible to find visibility networks in various works:

In [13] the authors demonstrated how to construct a network from a time series of United States (US) hurricane counts and show how it can be used to identify unusual years in the record, based on a “line of sight” visibility algorithm. Another example is [32], where the authors analyzed the gold price time series by mapping the time series into a visibility graph network analyzed in order to extract trends in the times series. For what concerns using visibility networks with the aim of depicting lines of sights among individuals in a specific environment, to the best of our knowledge, we were not able to find any similar work.

Chapter 3

Background

Each implemented step of the framework relies on various data structure, procedures and basic concepts combined with the aim of extracting more information from the input data. In order to offer a complete and clear understanding of the whole process, in this chapter we will go through the concepts and methods on which the work done in this thesis is based and further developed.

3.1 Tree Data Structures

In order to store the data we have processed, we used tree-like data structures.

A tree is defined as a data structure composed by nodes and edges, not containing any cycle. A non empty tree is made of a root node and many depth levels containing other nodes that all together constitutes a hierarchy.

In this thesis we focus our attention on two kind of trees data structure, used in order to reach the final objective, which are the Binary search Tree (BST) and the K-Dimensional Tree (KDT).

3.1.1 BST

The Binary Search Tree is a tree-like data structure which respects the property that the key in each node must be greater than all the ones stored in the sub tree on the left and smaller than all the keys in the subtree on the right [10]. The major advantage of this data structure is that search, insertion and deletion of elements can be done with an average time complexity of $log(n)$ by leveraging on the specific ordering of the elements in the tree. In Figure 3.1 we report an example of BST.

3.1.2 KDT

At a high level, a KDT is a generalization of a BST that stores points in k-dimensional space [5]. An example of KDT is reported in Figure 3.2.

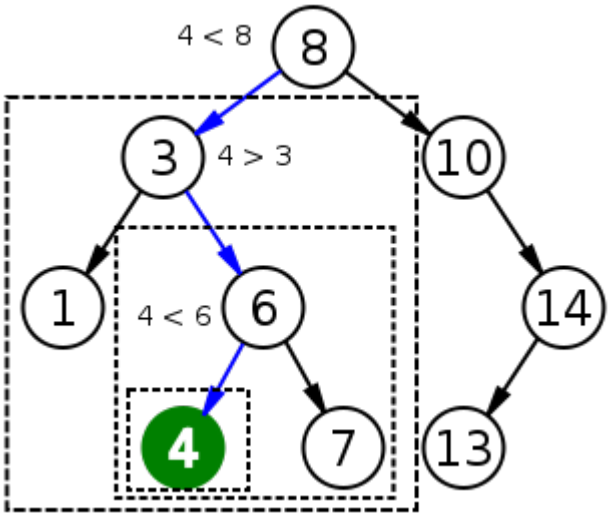

Figure 3.1: An example of BST.

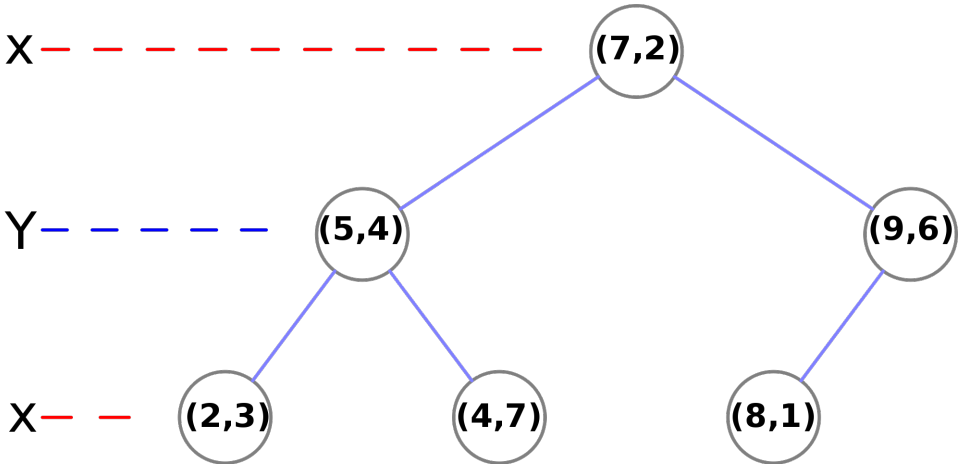

Figure 3.2: An example of KDT.

As we can see, a KDT is a k-dimensional binary search tree, where each point defines an hyperplane dividing the k-dimensional space in two parts, where the left part is represented by the left subtree and right part from the right one. The greatest advantage of this separation is given by the speed up obtained in the nearest neighbors lookup, which is performed through the following procedure:

I) Take a guess of where the nearest neighbor is located by taking all the points inside a certain hypersphere of radius r.

II) Determine if the candidate hypersphere crosses one of the splitting hyperplanes by checking if $|b_i - a_i| < r$.

III) If the hypersphere is just on one side of the hyperplane, then we need to check only that side, otherwise recursively search both sides of subtrees corresponding to the two parts of the hyperplane.

This procedure yields an average time complexity of $m \cdot log(n)$, where m is the number of nearest neighbors that we are seeking and n the number of elements in the tree. The speed up is given by the fact that we are guessing where the nearest neighbor is and that our hyperplanes are all axis-aligned, making checking if they an hypersphere crosses them or not a feasible task, thus guiding towards the right place where to search in a fast and simple way.

In this thesis we used the KDT structure in order to perform fast lookup of the neighbors of a specific point in the point cloud.

3.2 Learning Methods

In machine learning, with the term learning we are referring to the procedure used to infer new information from what it is already known about a certain dataset. Learning can be achieved in two main ways, known as *supervised learning* and *unsupervised learning*.

Supervised learning is defined as the procedure used to learn a function f such that given an input x, corresponding to a set of characteristics of a specific element, the function is able to infer from x the unknown class y to which the the element belongs, meaning that $f(x) = y$. Unsupervised learning instead consists in looking for meaningful patterns in the data without having any prior knowledge about the class, thus it is more exploration than inference oriented.

In the next section we will see in details the supervised and unsupervised learning methods, given the fact that they have both been used in this thesis.

3.2.1 Classification

The most common supervised learning method is classification, which consists in training a classifier on some labeled data (for which the classes to which data belong

are known) in such a way that, when tested on unseen data, the classifier will predict the right label for those data, given their features [37]. A feature is defined as a measurable property of a certain observed phenomenon. The process consists in two main phases: training and testing. During training, a certain amount of labeled data is used to learn useful patterns with respect to the class label and the features. During testing instead, the algorithm is used to infer the label of previously unseen data by means of the patterns detected during the training phase. Given that labels of data used to train the algorithm are known, procedures computing error measures by comparing the label inferred by the algorithm and the real one could be easily applied. Data used to make our classifier learn interesting patterns are called *training set*, while the ones used to test our classifier and verify whether the predictions obtained are correct or not are called *testing set*. Sometimes, when there is the need to tune some parameter of the algorithm, another set of data, known as *validation set* is used to find the best value for the parameters. A necessary condition is that training and testing set must be *i.i.d.*, given that any kind of correlation between the two would lead to biased predictions.

In this thesis we applied this approach to infer the visual obstruction potential of vegetation elements in the point cloud. In the next sections we will go into details of the supervised classification algorithms we used to perform inference.

Decision Trees

A decision tree is a tree-like structure in which internal nodes represents a check on the value of a certain feature (a certain characteristic of an element), each branch represents one of the outcomes of the previous check (what we wanted to verify) and each leaf node represents a specific class (final value assigned to the specific element) [37]. Each paths from root to leaf is a classification rule, which basically corresponds to a set of values over the checks performed at each internal node which guide toward a specific class. The tree structure is learned maximizing the *information gain* at every possible split.

Formally, let $T = \{x_1, ..., x_n, y\}$ be a set of training examples where x_i is the i-th feature and y is the class label, then, for a certain feature i, information gain is defined as:

$$I(T, i) = H(T) - \sum_{v \in vals(i)} \frac{|\{x \in T | x_i = v\}|}{|T|} \cdot H(\{|\{x \in T | x_i = v\}, \tag{3.1}$$

where H represents the *entropy*, which, given a discrete random variable X with values $x_1, ..., x_n$ and probability mass function $P(X)$, is defined as:

$$H(T) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i) \tag{3.2}$$

An example of decision tree to decide whether it is safe to play outside or not based on weather features is reported in Figure 3.3.

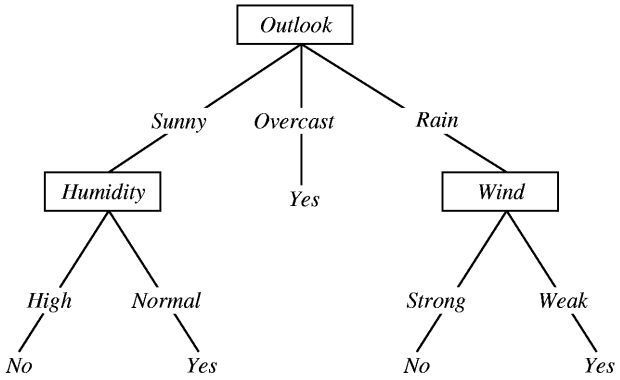

Figure 3.3: An example of Decision Tree to decide whether it is safe to play outside or not based on weather features.

K-Nearest Neighbors (KNN)

KNN is a memory based supervised classification algorithm, meaning that previously unseen data are classified with the class of the least distant element or elements in the training set, according to the k chosen [37]. In order to make the algorithm work properly it is necessary to define a suitable distance measure between data instances. The k parameter is used to determine how many neighbors must be considered in distance order. Once identified the k closest neighbors, the final prediction is obtained by performing majority voting on the classes of found neighbors. In case of a tie, a random decision is made. It is immediate to notice that the choice of the k parameter and the distance measure are the crucial factors in such approach. An example of KNN is reported in Figure 3.4.

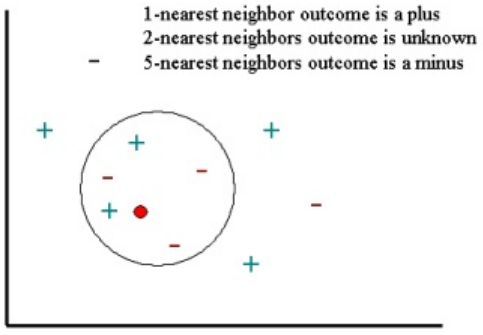

Figure 3.4: An example of KNN where the target point is the one in red.

Naive Bayes

The naive bayes classifier is a supervised classification method using probabilities to perform classification, based on the assumption that all the features in the data are independent from the class [37]. By exploiting to this assumption, the conditional probability of a training instance $x_1, ..., x_n, y$, where $x_i \in X_j$ is a value of the j-th feature and $y \in Y$ is the class label, could be rewritten in the following way:

$$P(X_1 = x_1, ..., X_n = x_n | Y = y) = P(X_1 = x_1 | Y = y) \cdot ... \cdot P(X_n = x_n | Y = y) \tag{3.3}$$

During the training phase all the conditional probabilities of the features values conditioned by every class label are computed from the training set. During the test phase instead, given a previously unseen instance $x_1, ..., x_n$, the predicted class label is the $\hat{y} \in Y$ that maximizes the following conditional probability $P(X_1 = x_1, ..., X_n = x_n | Y = \hat{y}$ computed using the probabilities estimated during the training phase.

Logistic Regression

Logistic regression is a statistical model that measures the relationship between the class label and one or more independent features by estimating probabilities by means of the *logit function* [37]. The *logit function* is reported in Figure 3.5.

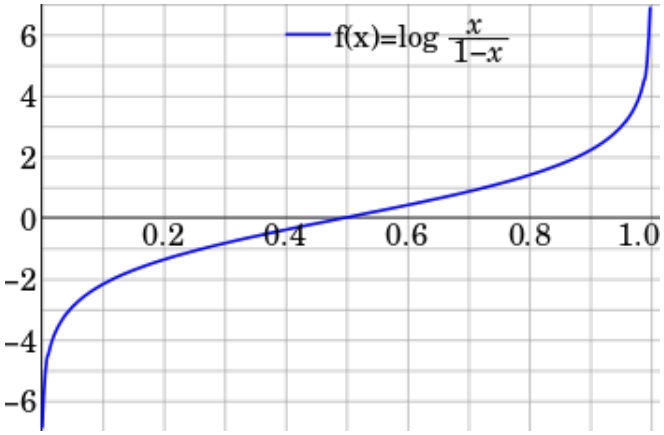

Figure 3.5: Logit function.

Formalizing, given a previously unseen instance $x_1, ..., x_n$ where $x_i \in X_j$ represents a value of the j-th feature, the probability of a certain class label is estimated in the following way:

$$P(Y = y) = \frac{e^{w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n}}{1 + e^{w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n}}, \tag{3.4}$$

where $w_0, w_1, ..., w_n$ are the features weight estimated during the training phase by minimizing a logarithmic cost function that penalizes in a logarithmic way each wrong prediction on the training set.

Random Forests

Random forest is an supervised classification ensemble method [37]. The term *ensemble* means that we train the same classifier a certain number of times, each time with different features, and then combine their outcome in some specific way in order to obtain the final prediction.

More in details, the random forest method consists in training an high number of decision trees, each one having as features a random selection of all the ones available in the training set. At testing time the final prediction is obtained by performing majority voting on all the outcomes of the built trees for the specific features' values of the tested data instance. The structure of the Random Forest classifier is reported in Figure 3.6.

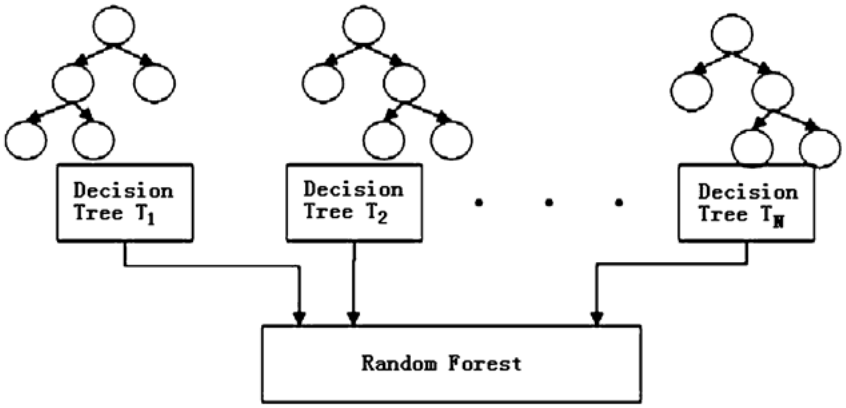

Figure 3.6: Structure of the Random Forest classifier.

Support Vector Machines (SVM)

SVM is a supervised learning model which locate the training instances into an high dimensional feature space, meaning that each feature becomes a dimension, and search for the best splitting hyperplane [37]. A best splitting hyperplane is defined as an hyperplane which separates the instances belonging to one class from the ones belonging to others classes with the best possible margin, meaning that the distance of points belonging to a certain class from the hyperplane is maximum. Points which have minimum distance from the hyperplane are called *support vectors*, given that moving those points would result in a different optimal hyperplane. When the number of dimension is too high, SVM exploits the *kernel trick*, leveraging on *kernel methods*, in order to avoid computing directly the new feature space with the aim of

lowering the number of dimensions. The explicit computation of the feature space is avoided by computing the vectorial product between two elements instead of the whole space transformation. An example of SVM applied to points in the 2D space is reported in Figure 3.7.

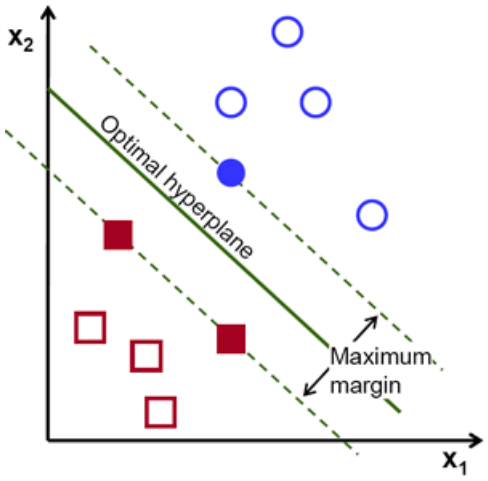

Figure 3.7: Example of binary classification using SVM where support vectors are represented by full points.

3.2.2 Clustering

The most common unsupervised learning method is clustering, consisting in grouping elements into clusters, where with the term cluster we are referring to a group of elements which are really similar to each other, according to a chosen similarity measure [18]. Elements belonging to different clusters instead generally are different from each other, always according to the chosen similarity measure. Once a cluster is identified, the *centroid*, which is the center of a cluster, is identified by the average value of each coordinate of the points in the cluster. The crucial aspects in such procedure are determining the best similarity measure with respect to data and objectives, together with choosing the optimal number of clusters in the data. An example of the clustering method is reported in 3.8.

In this thesis we have used an approach similar to clustering in order to identify points in the point cloud corresponding to a specific tree, leveraging on the fact that distances between points belonging to the same tree are generally different from distances between points belonging to different trees.

3.3 Kernel Methods

Given that we made use of algorithms exploiting kernel methods, such as SVM, we briefly describe what they are and how they work.

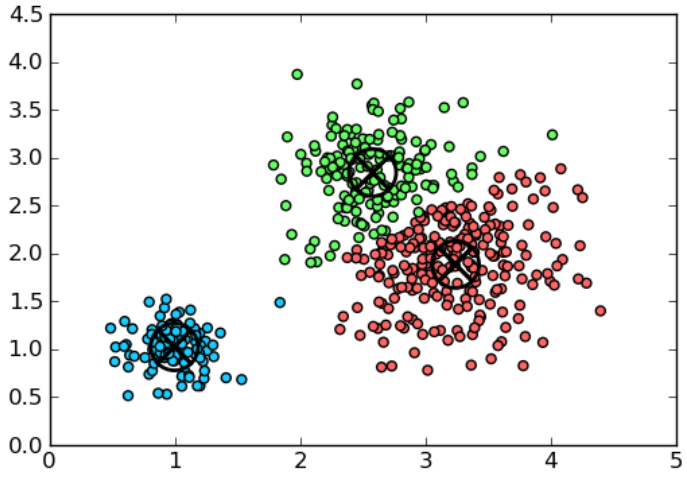

Figure 3.8: Example of clustering where points are grouped in three main clusters and centroids are represented by crosses inscribed in a circle.

In machine learning kernel methods are techniques which allow to perform a mapping between two different feature spaces without explicitly computing the spaces, thus offering a great computational advantage [1]. Kernel methods are mostly used in pattern analysis. The general aim of pattern analysis is to determine a function that relates the input data with the output ones. The great advantage in using them is that it not required to transform the data in raw representation into a feature vector representation, given a specific mapping defined by the user. Instead it suffices to define a user-specified kernel, which is a similarity function over pairs of data instances in their raw representation.

In the next sections we will go through details of the specific kernel methods used in this thesis.

3.3.1 Density Kernel

Let $x_1, ..., x_n$ be an identically independently distributed (i.i.d.) sample taken from a distribution with unknown density f. The objective of the density kernel is estimating the shape of f by using the following kernel density estimator [43]:

$$\hat{f}_h = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h}, \tag{3.5}$$

where K is a kernel function, which is non negative, integrates to one and has zero mean. The h parameter is a smoothing parameter, also known as ***bandwidth***, which must be greater than zero. $K_h(x) = \frac{1}{h}K(\frac{x}{h}$ is called scaled kernel with subscript h.

In this thesis we used the density kernel to estimate the density of the points corresponding to a specific tree.

3.3.2 Linear Kernel

In mathematics a linear kernel is defined as a linear mapping between two vector spaces [1]. Formally, let V and W be two different vector spaces, L be a linear mapping between V and W and v be an element of V, then a linear kernel is defined in :

$$ker(L) = \{v \in V | L(v) = 0\} \tag{3.6}$$

In this thesis we used the linear kernel with SVM to apply the kernel trick.

3.3.3 Polynomial Kernel

The polynomial kernel is used to represent the similarity of vectors over polynomials, which are derived from the original variables, thus allowing the use of non-linear models [7]. Given a degree *d* polynomial and two input vectors *x* and *y* in their original feature space, the polynomial kernel is defined as:

$$K(x, y) = (x^T y + c)^d, \tag{3.7}$$

where *c* is a free-parameter used to balance the influence of higher-order terms with respect to lower-order ones in the polynomial.

For what concerns the work done in this thesis, the polynomial kernel is used with SVM to perform inference exploiting the kernel trick.

3.3.4 Averaging Kernel

The averaging kernel is a $N \times N$ sliding-window spatial filter, which assigns to the central value of the $N \times N$ matrix in input the average of all points in the matrix [25]. The matrix in input is referred to as *window* or *kernel*.

The averaging kernel is used during the *multiscale curvature algorithm* used to identify ground and non-ground points in the point cloud surface.

3.4 Point Cloud Representation

The analysis on the forested surfaces used in this thesis were performed on the point cloud representation of the surfaces. A point cloud representation is a collection of points bringing 3D information relative to an object or a surface [44]. It is a very basic discrete representation, essentially specifying the geometry of the object by means of sampling procedures at certain positions. In the geographic field they are

mostly used in order to create digital elevation models of the terrain. An example of point cloud surface is reported in 3.9.

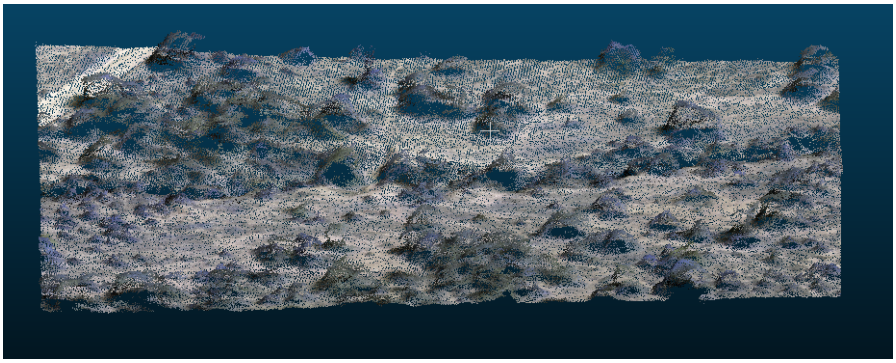

Figure 3.9: Example of point cloud surface.

3.5 Probability Distributions

Many of the methods already seen leverage on the concept of probability distributions. Probability distributions are used to assign a certain probability value to each subset of a chosen random experiment [15]. There are two main types of probability distributions:
Discrete probability distributions, where it it possible to assign a probability to each possible outcome of the experiment, and *continuous* probability distributions, where random variables take value from a continuum, thus probabilities can be assigned to both intervals and individual values. Two are the main continuous probability distributions used in this thesis, which we are going to discuss in the next sections.

3.5.1 Gaussian Distribution

The Gaussian distribution is one of the main probability distribution widely used in various fields, given that a set of random variables can be approximated to a Gaussian distribution under the following conditions [15]:

- The random variables must be i.i.d.
- They must have a well-defined mean
- They must have a well-defined variance

The Gaussian distribution is defined by the following probability density function :

$$f(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.8}$$

where μ is the mean of the distribution and σ the variance.

3.5.2 Student t-distribution

The Student's t-distribution is a member of the family of continuous probability distributions, which are used in order to estimate the mean of a normally distributed population (it follows a Gaussian distribution) with small sample size and unknown standard deviation [15].

Formalizing, let v be the degrees of freedom of the distribution, which correspond to the number of variables in the distribution, then the Student t-distribution is defined by the following probability density function:

$$f(t) = \frac{\Gamma(\frac{v+1}{2}}{\sqrt{v\pi}\Gamma(\frac{v}{2}}(1 + \frac{t^2}{v}^{-\frac{v+1}{2}}, \tag{3.9}$$

where $\Gamma(t)$ is called the ***Gamma Function***, defined as

$$\Gamma(t) = \int_0^\infty x^{t-1}e^- x dx \tag{3.10}$$

3.6 Statistical Significance Tests

Once results of a prediction task are obtained, statistical significance tests are used in order to verify whether the results obtained are significant or not. Statistical significance tests consist in checking if a quantity known as ***p-value*** is less than a chosen significance level α [15]. The ***p-value*** is defined as the probability of obtaining at least as extreme results as the one observed, given that the ***null hypothesis*** is true. The null hypothesis is defined as the default condition according to which there is no relationships between the samples that we are considering. The significance level instead is defined as the probability of rejecting the null hypothesis given that it is true. When the ***p-value*** is less than α, we can say that our samples are statistically significant at a $1 - \alpha$ level. Statistical test are widely used in order to verify if the improvement in the prediction results obtained is significant with respect to to another result obtained using a different prediction technique, based on the fact that comparisons among different prediction results are meaningful only if the relative samples are significant. For what concerns the tests used in this thesis to verify the results of the inference task, we used the ***Student t-test*** and the ***The Analysis of Variance*** (ANOVA) test, explained in the following sections.

3.6.1 Student t-test

The Student t-test is a statistical test used to check hypothesis where the test samples are defined by a Student t-distribution, under the condition that that the null hypothesis is satisfied [15]. This test can be used to determine whether two samples are significantly different from each other given their means.

3.6.2 The ANOVA Test

The ANOVA test is a statistical significance test used to analyze the differences among groups means and variances [15]. More in details, the ANOVA test performs a statistical test with the aim of checking if the means of several groups are equal, thus it generalizes the t-test to more than just two groups.

3.7 Validation Metrics

Validation of results obtained through a classification process is a very important aspect, given that we want to be as sure as possible that the predictions made are reasonable and correct. Given that we want to properly validate the results obtained through supervised inference, we are going to discuss the validation metrics and procedures adopted. There are many techniques that can be used to perform validation, but before introducing them we first present the nomenclature used in their definition [37]:

1. **TP**: are instances which are predicted as positive (class represented as 1 in binary classification) according to the truth label and the classification outcome.

2. **TN**: are instances which are predicted as negative (class represented as 0 in binary classification) according to the truth label and the classification outcome.

3. **FP**: are instances which are predicted as positive by the classifier but they are negative according to the truth label.

4. **FN**: are instances which are predicted as negative by the classifier but they are positive according to the truth label.

These definitions are intuitively represented in the matrix reported in Figure 3.10, known as *confusion matrix*.

In the next sections we will define the most common validation measures and techniques used in the field.

3.7.1 Accuracy

According to [37], accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.11}$$

Accuracy measures the percentage of correctly classified instances with respect to all the instances in the test set.

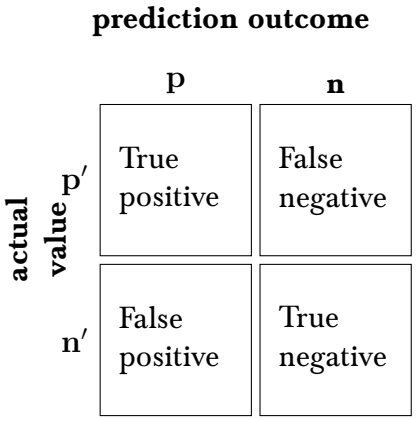

Figure 3.10: Confusion matrix.

3.7.2 Recall

According to [37], recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.12}$$

Recall measures the percentage of the positively instances correctly classified with respect to all the positive instances in the test set.

3.7.3 Precision

According to [37], precision is defined as:

$$\textit{Precision} = \frac{TP}{TP + FP} \tag{3.13}$$

Precision measures the percentage of the positively instances correctly classified with respect to all instances classified as positive.

3.7.4 Cross-Validation

Cross-validations is a model validation technique with the aim of testing how the results of a certain prediction will generalize with respect to an independent dataset [37]. Cross-validation consists in dividing the data available in a certain number of folds, all stratified (with the same number of positive and negative instances). Once created the folds, the model is trained on all the fold except one, which will be used as test data. This is done until all folds have been tested. The final error is given by the average of the errors computed on each fold. When each fold contains only one instance, then we are performing a validation technique called *leave one out cross-validation* (LOOCV), given that we train our model on all the instances available

except one. In LOOCV the error is computed as the number of correct predictions on all the folds divided by the number of folds, which corresponds to the size of the dataset.

3.8 Geographic Coordinates Systems

In this section we introduce the notion of geographic coordinates system, given that points in the point cloud are relative to specific geographic locations on Earth. A geographic coordinate system is a coordinate system that enables every location on the Earth to be specified by a set of numbers or letters, or symbols. The coordinates are often chosen such that one of the numbers represents vertical position, and two or three of the numbers represent horizontal position. A common choice of coordinates is latitude, longitude and elevation. In order to specify a location on a two-dimensional map a map projection is required.

In this thesis coordinates of points in the point cloud are expressed using the the Universal Transverse of Mercator (UTM) coordinates system, which we are going to describe in the following section.

3.8.1 UTM Coordinates System

The UTM coordinates system is a system which uses two coordinates to give locations on the surface of the Earth: the first coordinate is called *Easting* and the second one *Northing* [11]. As with latitude and longitude, it is a horizontal position representation, meaning that the location on Earth is identified without considering any vertical component representing the vertical position of the point. Differently from latitude and longitude, the UTM system cannot be considered a single map projection, given that it divides Earth in sixty zones, where each zone corresponds six-degree band of longitude.

3.9 Interpolation

Given that in this thesis we used the interpolation procedure to generate a new point cloud surface starting from the original one when performing the ground/non-ground separation of points, we now introduce the relative concepts. When we use the term interpolation we are referring to the generation of new data points in the range of a an available set of discrete data points [41]. An example of interpolation applied to mathematical functions is reported in Figure 3.11. For what concerns the work done in this thesis, we are mostly interested in the *Thin Plate Spline* (TPS) interpolation, presented in the next section.

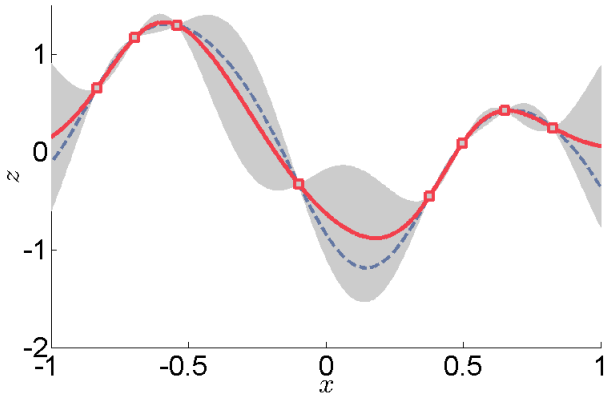

Figure 3.11: An example of interpolation of a curve (blue dashed) generating an approximate one (red).

3.9.1 TPS Interpolation

TPS interpolation is an interpolation technique that, given a set of data points, performs interpolation by means of a weighted combination of thin plate splines, which are numeric functions that are defined piecewise by polynomial functions [6]. Each spline is centered about each data point and generates the interpolation function passing through the points while at the same time minimizing the following integral:

$$I[f(x,y)] = \int\int_{R^2} (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2) dxdy, \tag{3.14}$$

where R^2 denotes the two dimensional space and f_{ab} denotes the second derivative computed first with respect to a and then b. The above integral is known with the name of *bending energy*.

3.10 Artificial Neural Networks (ANN)

In this section we introduce ANN, given the fact that they were used in this thesis to perform features extraction from the trees point structure in an automatic way. In machine learning ANN are models reproducing the biological neural networks and are used to approximate functions depending on generally a huge number of inputs [50]. An ANN can be depicted as a directed graphic model in which nodes of the graph are called *artificial neurons*, which exchange messages in a direct way with other neurons to which they are connected in the graph.

3.10.1 Artificial Neurons

An artificial neuron is a mathematical function built to model biological neurons. They receive one or more inputs (representing dendrites) and combine them to produce an output (representing a neuron's axon) [50]. The contribution of each node is weighted and the sum is propagated by means of a non-linear function, known as *activation function*. The first activation function used was sigmoid shaped (output between 0 and 1, defined for all real input values and has a positive derivative at each point), while in the last years non-linear functions have been widely used, such as the *hyperbolic tangent* (antisymmetric function with output between 1 and -1 and defined everywhere on the x-axis). The shape and formula of the sigmoid function are reported in Figure 3.12, while of the hyperbolic tangent in Figure 3.13. Activation functions must be monotonically increasing, continuous and preferably differentiable in order to easily compute the gradient.

Formalizing the definition of an artificial neuron, the output y is obtained through the following formula:

$$y = \phi(\sum_{j=0}^{m} w_j x_j + b), \tag{3.15}$$

where ϕ denotes the activation function, x_j and w_j the j-th input and its weight respectively and b the bias of the specific neuron. In Figure 3.14 we report the structure of an artificial neuron.

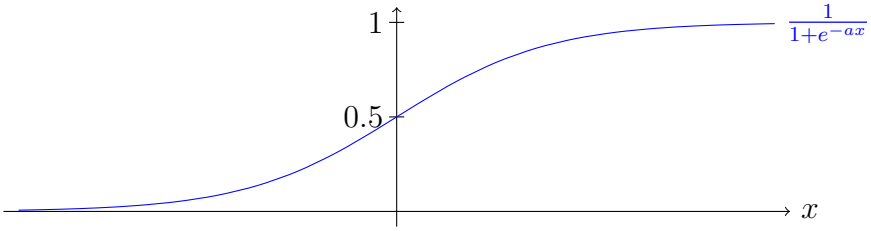

Figure 3.12: Sigmoid function.

3.10.2 Structure of ANN

The direct edges connecting nodes in the graph are called *connections*. Each connection between different neurons has a numeric weight, which is tuned according to experience, thus making neural networks a model adaptive to inputs and suitable for learning. Generally neurons are grouped in *layers*, which are level of the network corresponding to a certain number of neurons. The first layer is called *input layer*, the ones in the middle are called *hidden layers*, given that values in those layers are generally not given in output, while the last one is called *output layer* [50]. ANN propagating input in successive layers are known as *feed-forward ANN*. In Figure 3.15

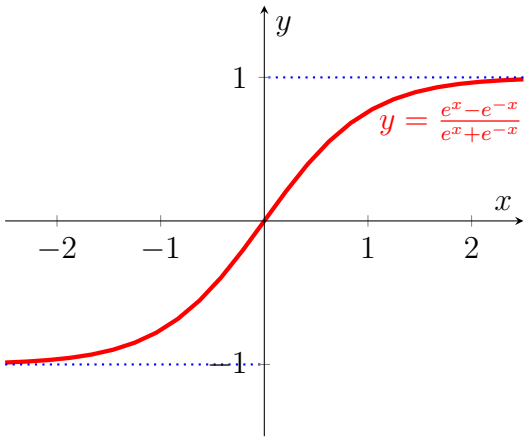

Figure 3.13: Hyperbolic tangent function.

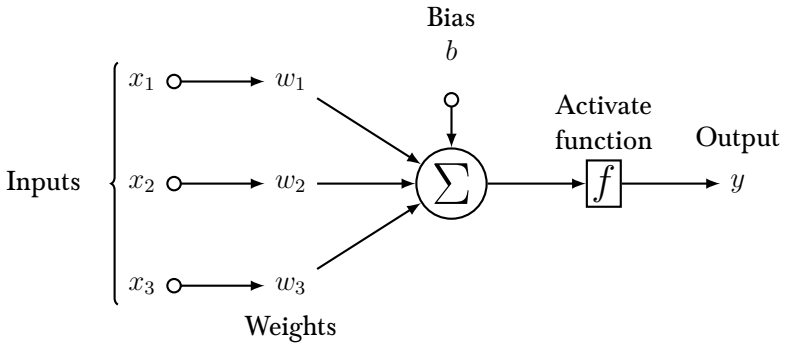

Figure 3.14: The structure of an artificial neuron.

we report an example of ANN with one hidden layer, five neurons in the input layer, three in the hidden layer and one in the output layer.

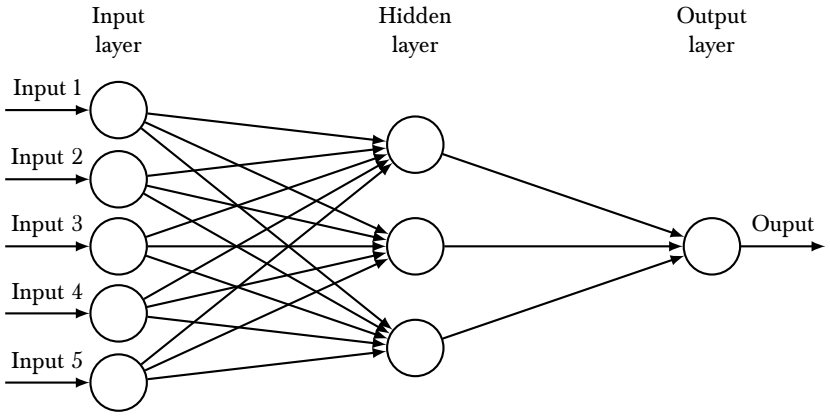

Figure 3.15: An example of ANN with one hidden layer.

3.10.3 Learning Phase

During the learning phase, the algorithm most commonly used is the *backpropagation algorithm* [42]. The algorithm consists of two phases, which we are going to describe in details through the steps performed at each phase:

- **Propagation**:
  1. Forward propagate the inputs in order to activate the successive layers until we obtain the output.
  2. Backward propagate the gradient of the cost function with respect to parameters in the network until reaching the input layer.
- **Weight Update**:
  for each connections the following steps are performed:
  1. Multiply the gradient computed during the propagation phase for a certain percentage, known as *learning rate*.
  2. Update the weight of the connection by subtracting to the old value of the one computed in the previous step.

The two phases are performed until we reach a certain desired error or for a certain number of cycles, also known as *epochs*. The learning rate greatly influences the speed and quality of learning, given that the greater is the ratio the faster neurons are trained, while the lower is the ratio the slower is the training.

3.10.4 Autoencoders

We now introduce a particular ANN of interest with respect to the feature extraction objective of this thesis. Autoencoders are an ANN mostly used to perform a *feature learning* task, which is the case of the work done in this thesis. With the term *feature learning* we are referring to the procedure in which we extract from raw input data interesting properties that could be used in a supervised classification task [4].

The difference between autoencoders and usual multilayer feedforward neural networks resides in the fact that autoencoders are trained to reconstruct the received input. The structure of an autoencoder is made of two main parts, *the encoder* and *the decoder*, which could be seen as two transition functions ϕ and υ:

$$\phi : \quad X \to F \tag{3.16}$$

$$\upsilon : \quad F \to X \tag{3.17}$$

$$argmin_{\phi,\upsilon} ||X - (\phi \circ \upsilon)X||^2, \tag{3.18}$$

where X represents the input space and F the feature space.

In the simplest case, which is when we have just one hidden layer, an autoencoder maps the input to a set of *latent variables* representing the initial input, where the adjective latent refers to the fact that these variables are estimated in the hidden layers of the network. Once the latent representation of the initial input x is generated, this representation is mapped to an output x' of the same shape of x, given the objective of reconstructing the initial input. In Figure 3.16 we report the structure of an autoencoder.

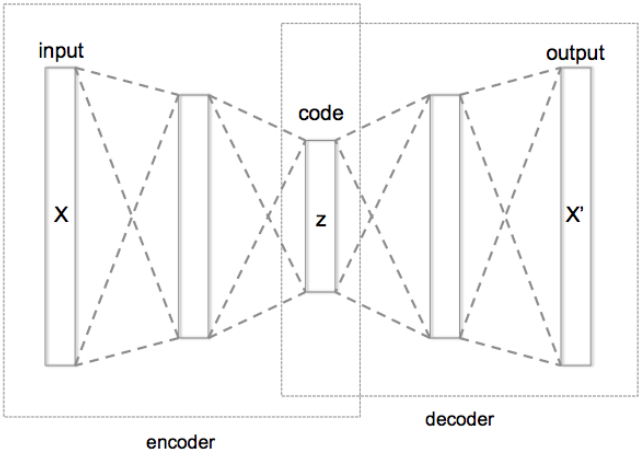

Figure 3.16: Structure of an autoencoder.

During the training phase of an autoencoder, for each input x, the following steps are performed:

- Compute activations in all hidden layers and in the end in the output layer in a feedforward way in order to obtain an output x'.
- Measure the deviation between x and x' by means of a chosen error metric.
- Backpropagate the error through the entire network and update the weights

These steps are performed for a chosen number of epochs, using at each epoch techniques aimed at avoiding overfitting. With the term overfitting we are referring to the situation in which our model will perform very good on the data used for training but poorly on the ones used for testing. Even if using backpropagation generally provides good results, complications may arise when training networks with many hidden layers. The reason why complication arises is the so called *gradient vanishing problem*, which occurs when errors are backpropagated to previous layers and the gradient gradually becomes smaller until its contribution is insignificant. This could probably lead to the network reconstructing the average of the input data, thus not updating the weights of the connections anymore, meaning that no useful information is extracted. The gradient vanishing problem is sometimes solved by means of more complex training techniques, event if they too often result in a very slow learning process and uneffective solutions, in particular with respect to the trade-off between training time and results, critical in neural networks. A reliable solution to this problem was proposed in [20], where the *pretraining technique* is introduced. This technique consists in using a set of initial weights that approximate the final result. More in details, the pretraining technique proposed in [20] involves training in sets of two neighboring layers the network and then fine-tuning the results using standard backpropagation. Different successful experiments using autoencoders to perform features extraction are discussed in [3, 30, 29].

3.10.5 Denoising Autoencoders

Denoising autoencoders are autoencoders that take a partially corrupted input (they introduce a noise component) while training with the aim of recovering the original undistorted input. This technique was introduced with an approach based on the concept of *good representation* [49].

A good representation is defined as a representation that can be obtained in a robust way starting from a corrupted input and thought to be of good use in recovering the corresponding original clean input. A necessary condition when corrupting the input is that the representation at a higher level must be stable and robust with respect to the input corruption itself. Furthermore, in order to train an autoencoder to denoise data, a preliminary stochastic mapping is necessary to perform $x \to \tilde{x}$ such that data are corrupted but $\tilde{x}$ can still be used as input for a standard autoencoder, thus reconstructing the initial input x.

3.10.6 Stacked Denoising Autoencoders

A stacked denoising autoencoder neural network is simply obtained by creating a deep neural network, meaning that we use an high number of hidden layers, where each hidden layer is a denoising autoencoder neural network. With this kind of structure the output of the first denoising autoencoder layer is forwarded to the successive denoising autoencoder layer until the output layers is reached. The structure of the stacked denoising autoencoder ANN is reported in Figure 3.17.

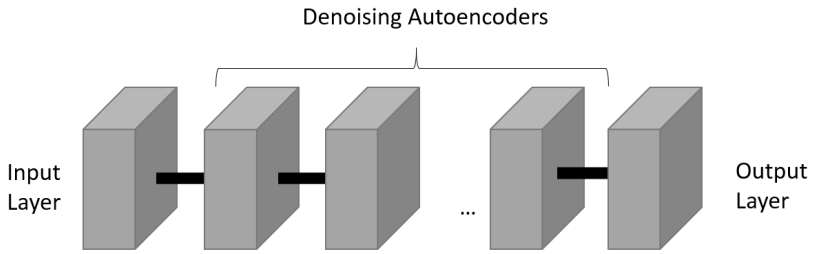

Figure 3.17: Structure of a stacked denoising autoencoder neural network.

Chapter 4

Dataset

Before going into details of how we reached the thesis objective, we first describe the data used to test the methods proposed in this thesis, their acquisition process and the relative ground truth together with its estimation process.

The point cloud data used are relative to many sites of the Mpala Research Centre in Laikipia region of Kenya (covering an area of 2.5 by 1.2 Km), where a drone *senseFly*[1] was deployed in order to properly capture the area. Drones are a very common technology used to capture surfaces, thanks to the information that could be extracted from *the ultra light images* of the surface [23]. The drone flew 11 days in January 2015, acquiring data on a different surface each day according to the drone range limitations. From each captured area, which was characterized by various photos taken at different height and from different angles, we obtained the point cloud 3D structure through storing the 3D structure as "las" [9] data format by means of the *senseFly* software on the drone. The derived point cloud has a resolution of 1.5 cm. However, there are two main issues related to the data acquisition method chosen. The first issue is that even if with various images different parts of the trees structure are properly captured, the inner parts of the same structure are often partially missing. Inner parts are missing because they are not properly detected due to the fact that they are covered by dense external parts of the same tree structure, thus not being captured by means of images only. The second issue is related to the fact that images capturing the surface included also shadows of objects. When combining the images into the 3D point cloud structure some assembled shadows formed amounts of points with low height but slightly over the ground, thus being classified as non-ground even if they actually are part of the ground.

For each point in the point cloud, the following features are available:

- **X**: coordinate corresponding to the *Easting* coordinate of the UTM system.
- **Y**: coordinate corresponding to the *Northing* coordinate of the UTM system.
- **Z**: coordinate representing the absolute height of points in the 3D space.

[1] More information on the company can be found at https://www.sensefly.com/home.html

- **Red**: red color component of the point.
- **Green**: green color component of the point.
- **Blue**: blue color component of the point.
- **Classification**: scalar field that is used by algorithms classifying points in ground and non-ground in order to mark them. According to the standards, a value of 2 for the classification field means that the point is part of the ground, while every other value different from 2 means that it is part of the non-ground elements.

In order to give a general idea of the forest environment in which we are going to perform the segmentation, Figure 4.1 shows an example of the Kenya savannah, which is the area captured by the photos with the aim of studying the behavior of baboons in their natural habitat.

As we can see in Figure 4.1, there is no spotted dominance of one kind of tree over the others, but instead there is a high variety of vegetation (trees and bushes of variable height). Due to the high variability in height, an effective segmentation algorithm should take vegetation variety into account by means of some flexible approach.

Figure 4.1: An example of the Kenya savannah vegetation.

4.1 Ground Truth Data

The areas that we are going to consider for the experiments are four, given the fact that in order to execute all the steps in the framework we must have the ground truth for the trees, thus having the possibility to verify the correctness of the results obtained.

The ground truth is available only for 119 trees, of which a measure of the height and diameter in two directions was taken, as well as notes about their species and conditions. A figure corresponding to the ground truth collection process is reported in Figure 4.2. By performing a visual analysis of the ground truth photos we estimated whether the corresponding trees could represent potential obstruction elements in the environment or not. Due to the fact that we need the ground truth in order to properly validate the results, we restricted the tests of the framework only to the four areas where the trees of which we have the ground truth are located.

Figure 4.2: Ground truth collection process.

4.2 Global Positioning System (GPS) Trajectories

For what concerns the construction of the visibility network, we used the GPS trajectories of a baboons troop living in the nearby areas of the Mpala Research Centre. The data were collected using GPS collars attached to each of the 26 baboons in the troop. In addition to collecting the Easting and Northing coordinates of baboons, the GPS collars report also the height of the individuals wearing it. The GPS position of a specific baboon was captured each second, for a total amount of 1505507 timestamps. However, due to many GPS errors in which the Easting or Northing coordinates are corrupted, a high percentage of the measurements is not usable as valid locations when performing the visibility analysis aimed at building the network.

Chapter 5

Ground/Non-Ground separation

The first operation that must be performed in order to reach the final objective proposed in this thesis is identifying which points in the point cloud correspond to the ground elements and which to the non-ground ones. In this chapter we will first go through the details of the algorithm used to distinguish between *ground* and *non-ground* points and its tuning procedure, then we will look at the results obtained using the algorithm with the tuned parameters on the areas of interest.

The algorithm used is the *multiscale curvature algorithm for Point Cloud Data* proposed in [14], which was implemented in the C++ freeware command line tool *MCC-LIDAR*[1]. The program requires in input two parameters:

- **Scale (post spacing)**: cell resolution of the surface given in input to the program.
- **Curvature threshold**: distance expressed in meters used to compare the interpolated surface with the original one.

In the following sections we will provide a detailed description of the multiscale curvature algorithm and a sequence of all the operations performed by means of suitable pseudo-code.

5.1 Method Description

The first step performed by the algorithm is looking for points with the same x and y coordinates, classifying them as non-ground and then proceeding to their removal from the initial point cloud U_0. This is a reasonable choice, considering that the interpolation of a raster surface requires that no two points have the same x and y coordinates. The reason behind the last requirement is that if points at the same location have the same value, they are seen duplicates and have they will not affect the output. If they have different values instead, they are seen as coincident.

[1]The software can be found at https://sourceforge.net/p/mcclidar/wiki/Home/

Interpolation functions could handle this kind of data in different ways, using in some settings the first point of the coincident points for the calculation, while in other settings the last. Because of this ambiguity, some locations in the output raster might have a value which is different from the expected one. This issue could be easily solved by properly preparing the data by means of the removal of these coincident points, as done in the first step of the algorithm. About the choice of classifying those points as non-ground, it suffices to notice that if two or more points have the same x and y location, then all the points except for the lowest point (i.e. the one with the minimum z coordinate) must be non-ground.

Once dealt with the coordinates issue, for three different scale domains (different surface resolutions), the following steps are performed:

I) A new raster surface is interpolated using the TPS interpolation technique [6]. Using TPS interpolation offers the possibility to adjust tension between points, fitting input data and handling the distance at which point samples influence the estimate of the surface [14].

II) A 3x3 averaging kernel is applied on the computed surface in order to regularize the interpolated surface.

III) For each point, if its height in the original surface is greater than the one in the interpolated surface plus a curvature tolerance threshold for the specific scale domain, then it's classified as non-ground and removed from the original point cloud.

For each scale domain, these steps are performed until we reach a certain convergence threshold. According to the experiments performed in [14], the best convergence criteria was continuing iterating until the number of classified points during the current loop iteration is less than 0.1% of the points still unclassified in the original point cloud. Once processed the surface for the three different scale domains chosen, the points which still remain unclassified are all classified as ground.

5.2 Pseudo-code

In this subsection we first introduce the nomenclature that we are going to use in the pseudo-code and then the detailed pseudcode of the algorithm:

- **SD**: scale domain, integer contained in [1,3] which represents the resolution at which the surface is scanned at the current iteration
- $t_{i \in SD}$: curvature threshold for scale domain i, contained in the set of scale domains used. The algorithm scans the surface with three different scale domains, incrementing t of 0.1 meters every time that we increase the scale domain, starting from the initial value given in input by the user.

- $\boldsymbol{CR_{i \in SD}}$: post spacing for scale domain *i*, contained in the set of scale domains used. For the first scale domain, the algorithm uses half of the value given in input by the user, for the second the exact value received in input and for the last the initial value incremented of half its value.
- $\boldsymbol{f}$: tension parameter (invariant across scale domains). The value assigned is 1.5, according to the analysis performed in [14].
- $\boldsymbol{U}$: vector of points that remain unclassified ($P_1, ..., P_n$).
- $\boldsymbol{n}$: number of points in U (at the beginning of each loop iteration).
- $\boldsymbol{P_j}$: single point in the cloud having coordinates x_j, y_j, z_j.
- $\boldsymbol{U_0}$: initial point cloud given in input by the user.

Once we introduced the nomenclature used, we report the pseudo-code of the main algorithm procedure in Algorithm 1 while the one of the subroutine handling points with same x and y is reported in Algorithm 2.

Algorithm 1 Multiscale Curvature Algorithm

```
procedure MultiScaleCurvatureAlgorithm
    U ← filterOutPointsSameXY(U_0)
    for scale domain SD = 1 to 3 do
        repeat
            S ← interpolate new raster surface using TPS(U, CR_SD, f)
            S' ← surface resulting from passing 3x3 averaging kernel over S
            for each P_j ∈ P do
                if z_j > S'(x_j, y_j) + t_SD then
                    classify P_j as non-ground and remove it from U
                n_C ← number of points classified and removed from U during
                        current iteration of the current loop
        until n_C < 0.1% · n
    classify all the points remaining in U as ground
```

For the ground/non-ground classification step we adopted the LIDAR classification standard, meaning that when a point is classified as non-ground the value of its classification field is set to any value different from 2 while when a point is classified as ground the classification field is set to 2.

5.3 Scale and Curvature Threshold Tuning

In order to assign the best values to the curvature threshold and the scale parameters, a few experiments with different settings were necessary. More in details, for

Algorithm 2 Procedure to filter points with same X and Y

```
1: procedure FilterOutSameXY(U_0)
2:     for each x,y location in U_0 with two or more points do
3:         Z_lowest ← minimum z coordinate of the points at x,y
4:         for each P_j at x,y do
5:             if z_j > z_lowest then
6:                 classify P_j as non-ground and remove it from U_0
```

the scale parameter we performed experiments using values in the range of $[0.5, 2.0]$ meters with an increment of 0.5 meters at each experiment but none of the results obtained improved the classification outcome with respect to the result obtained using the default value, which is 1.5 meters. For this reason we tuned only the curvature threshold parameter and left the other fixed at 1.5 meters, value suggested for this parameter by the developers of the application on the software page. The results of the performed tuning experiments for the curvature threshold are shown in figs. 5.1 and 5.2.

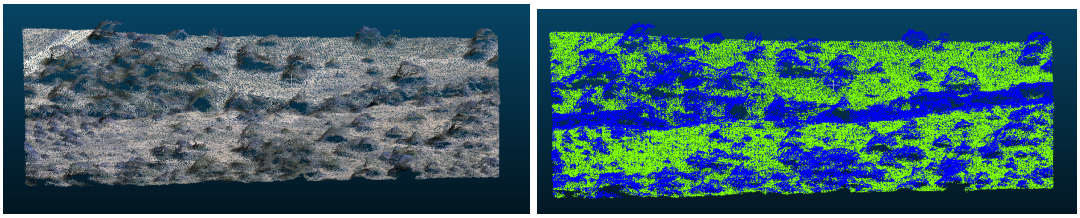

Figure 5.1: Example of a point cloud surface corresponding to a Kenya surface (on the left) and the same surface classified using a curvature threshold of 0.1 (on the right) where vegetation points are highlighted in blue.

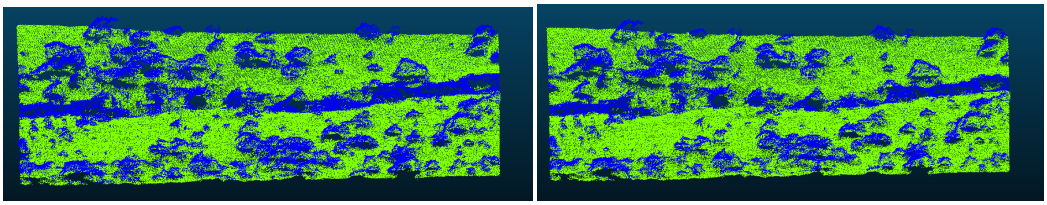

Figure 5.2: Surface in 5.1 classified using a curvature threshold of 0.2 (on the left) and a curvature threshold of 0.3 (on the right) where vegetation points are highlighted in blue.

During the analysis of the results shown in figs. 5.1 and 5.2 by means of *CloudCompare*[2], a software that allows to explore the point cloud by zooming and rotating it, we discovered that a curvature threshold of 0.1 would result in classifying many

[2]The software can be found at http://www.danielgm.net/cc/

ground points as vegetation (many points of the road in the upper left corner of the surface), while using a curvature threshold of 0.3 would classify many lower trees as ground. Given that 0.2 represents a good compromise between overclassifying and underclassifying, we chose it as best value for the algorithm parameter instead of 0.3, which is the value suggested by the developers of the application for the curvature threshold parameter.

5.4 Separation Results

In this section we show the results of the ground/non-ground separation of the points in the point cloud for the four areas of interest. The results of the experiments on the areas of interest are reported in figs. 5.3 to 5.10.

By looking at the results shown in figs. 5.3 to 5.10 we can easily notice that almost all the vegetation points are correctly classified but the ground/non-ground separation procedure introduced a strong noise component, partially due to the shadows in the images used to build the point cloud, which were often classified as non-ground elements. In order to weaken the noise component we used the *LasTools*[3] software suite, which contains a set of tools implemented in *C++* aimed at processing "las" files. More in details, we used the *"lasnoise"* tool, which scans the point cloud data looking for isolated or sparse points and remove them from the point cloud. By using this software, thanks to the fact that the points relative to shadows were not much dense and relatively sparse, we were able to remove a great percentage of noisy point from the classified point cloud. A light noise component still remains after the noise removal operation but this does not represent a crucial issue, given that noisy points are sparse and isolated, thus they can be removed by further noise filtering operations during the segmentation phase.

[3]The software can be found at http://www.cs.unc.edu/ isenburg/lastools/

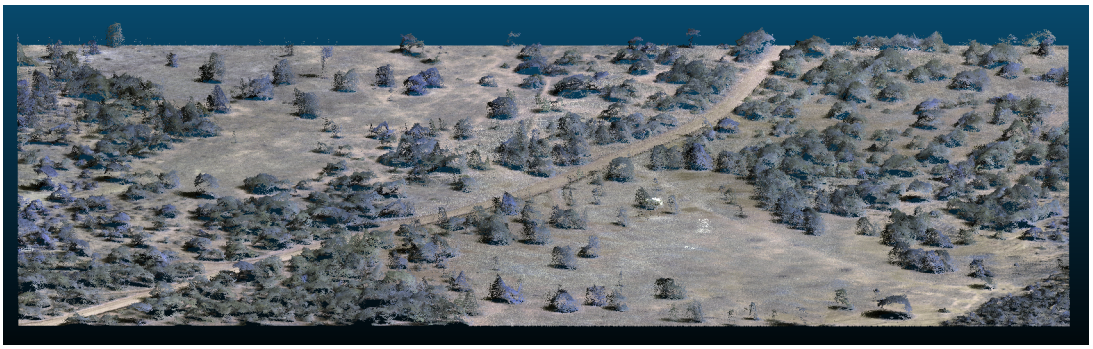

Figure 5.3: Area 1 of the ones of interest.

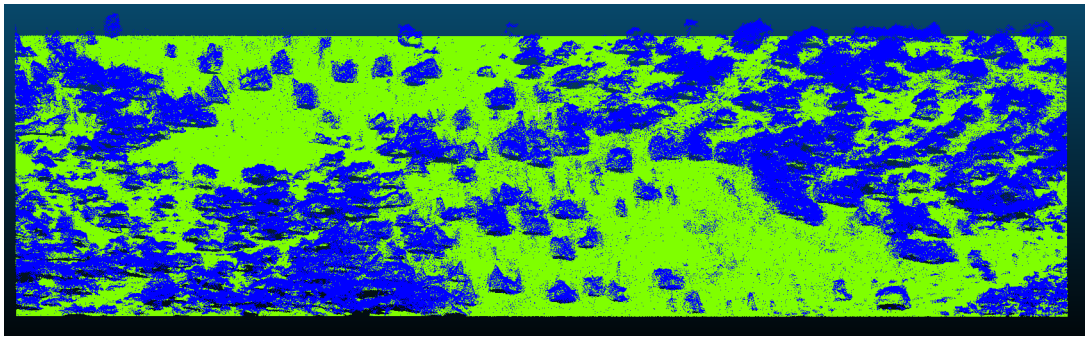

Figure 5.4: Area 1 of the ones of interest classified (the blue points correspond to the vegetation).

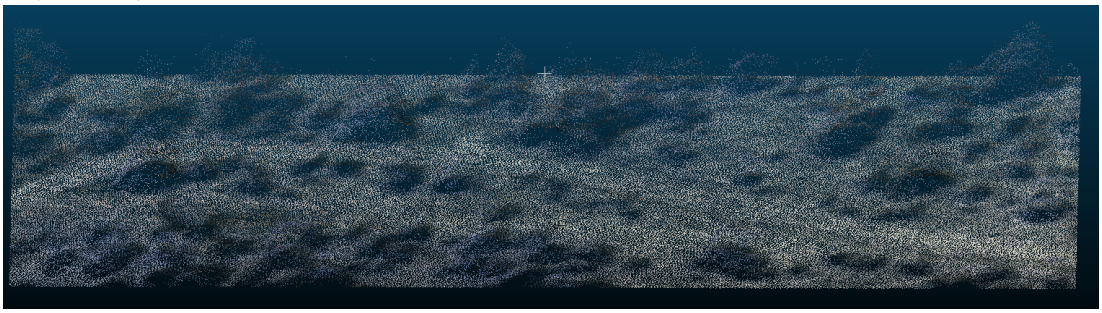

Figure 5.5: Area 2 of the ones of interest.

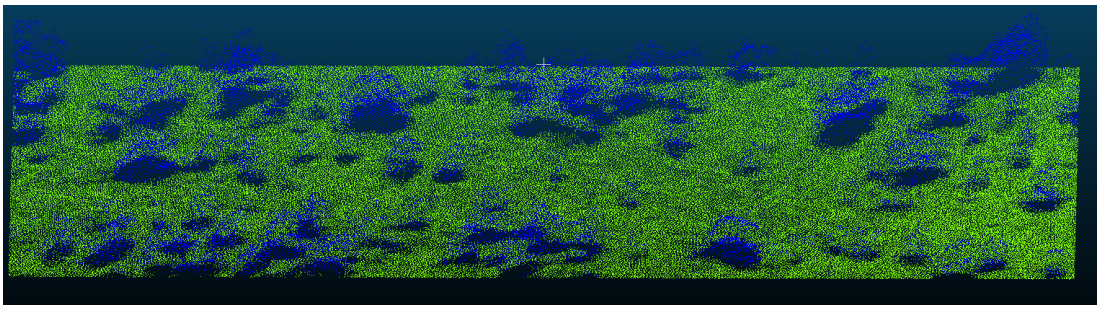

Figure 5.6: Area 2 of the ones of interest classified (the blue points correspond to the vegetation).

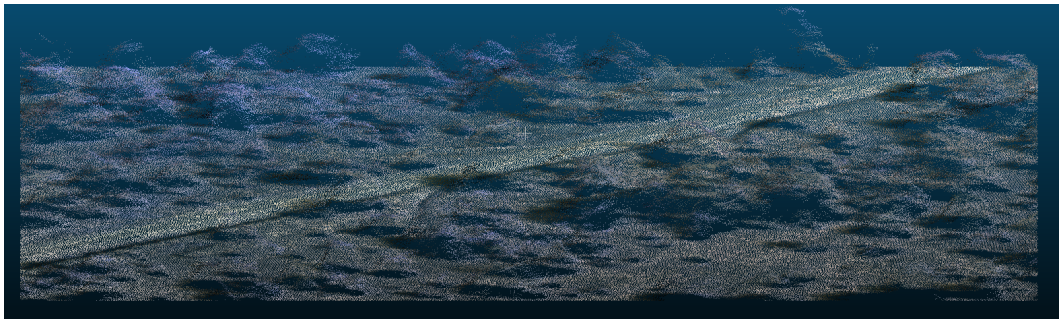

Figure 5.7: Area 3 of the ones of interest.

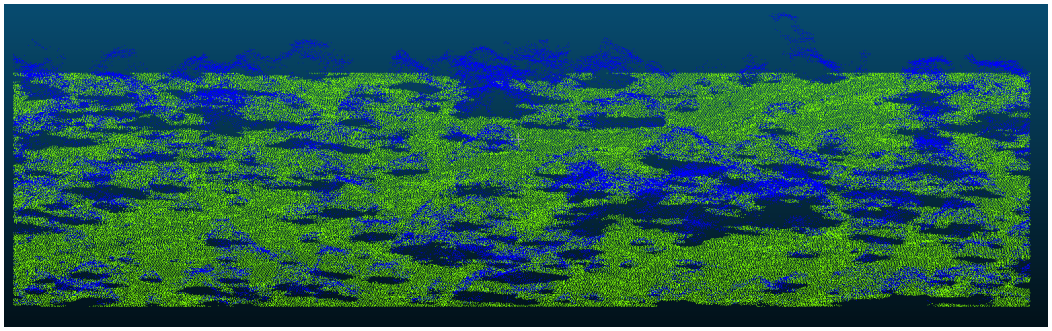

Figure 5.8: Area 3 of the ones of interest classified (the blue points correspond to the vegetation).

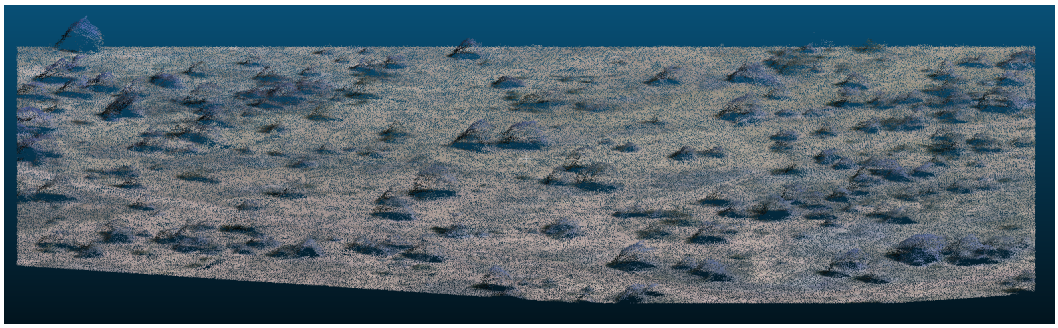

Figure 5.9: Area 4 of the ones of interest.

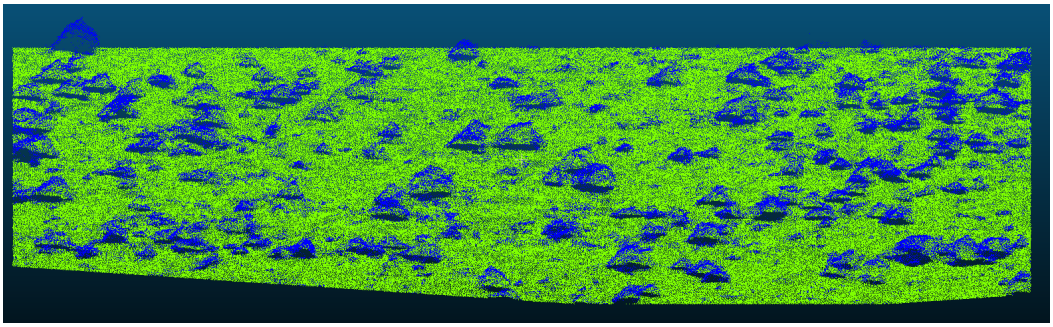

Figure 5.10: Area 4 of the ones of interest classified (the blue points correspond to the vegetation).

Chapter 6

Tree Segmentation

With the information on which points belong to the ground and which to the non-ground, we can proceed with applying an effective procedure aimed at identifying individual vegetation elements among all vegetation points identified in the previous phase. In this chapter we will go into details of the *segmentation algorithm* used on vegetation. After having classified the points into ground and non-ground ones, we first computed the real height of the vegetation points with respect to the ground ones in order to remove noisy points with negative or excessively positive height and level the ground. The ground was leveled using the *"lasheight"* tool of the *lasTools* suite, which uses the points classified as part of the ground to compute a *Triangular Irregular Network* (TIN) and then calculates the elevation of each point with respect to the computed TIN. Once we performed this operation, we applied the segmentation algorithm to the surface where height of vegetation points is now expressed with respect to the previously identified ground.

In the following sections we will provide a detailed description of the segmentation algorithm used and of the reasons behind its implementation. We will also provide a sequence of all the operations performed by the algorithm by means of suitable pseudo-code.

6.1 Method Description

The proposed method processes the whole cloud finding one tree at each iteration, as in [26]. For the segmentation of each tree the main idea is separating the points into two groups, the ones which actually belong to the tree structure and the ones belonging to other trees. Once we completed the process for one tree, the points corresponding to it are removed from the point cloud and the process continue until there are no points left in the initial point cloud. In order to speed up the process, we avoided considering the whole cloud at every iteration, but instead we picked a certain ball of points around the tree top of the considered tree, such to include all the points belonging to it. The estimation of the right radius requires a trial and

error process, given that for each kind of forest it is necessary to estimate how many points can belong to a certain tree at maximum in order to avoid losing part of the tree structure. In order to find the target tree top, we look for the highest point in the cloud, which represents the top of the tree that we want to identify during this iteration.

The next step performed is getting all the neighbors' points around the tree top within a chosen radius such to consider the whole tree structure. Once we identified this subset, we keep scanning the point from top to bottom and assign each point to one of the groups according to the minimum distance of each candidate point with respect to the two groups and to an adaptive distance threshold. Proceeding from top to bottom is the best way to identify the candidate tree, given that trees are better delineated at higher levels, while close to the bottom it is hard to determine to which tree the points belong, given possible overlapping among part of the trees. This is the reason why adopting a top down scanning direction would also ease the analysis of lower points, given that they will be analyzed once that the tree points structure is already partially identified.

Once we identified the subset relative to the target tree top, the algorithm is initialized by putting in the group of points relative to the target tree the found tree top, while in the group of the discarded points we put the point in the subset with greatest 3D euclidian distance from the target tree top. After the initialization step , each point in the subset is assigned to the right group by means of the following checks:

I) First we check if the minimum distance of the candidate point from the points in the cluster is greater than an *adaptive distance threshold*, determined according to the height of the candidate point. In case of a positive result for this check, we add the point to the discarded ones, otherwise we proceed to step **II**.

II) We check if the minimum distance of the candidate point from the points in the cluster is smaller than the minimum distance of the candidate point from the points discarded during the segmentation of the tree. In case of a positive outcome, the point is added to the cluster points, otherwise it is added to the discarded ones.

We remind that the scanning direction for the points is from top to bottom. Every time that a point is added to one of the two groups, we remove the point from the subset that we are considering, so that by taking the maximum each time we are automatically scanning the points left in the subset in height order. The set of operations performed to assign each point in the subset into the right group is summarized in the flow chart in 6.1.

From the analysis of the clustering rules listed above, it is clear that the choice of the distance threshold is the critical one. An extensive search of the parameter space

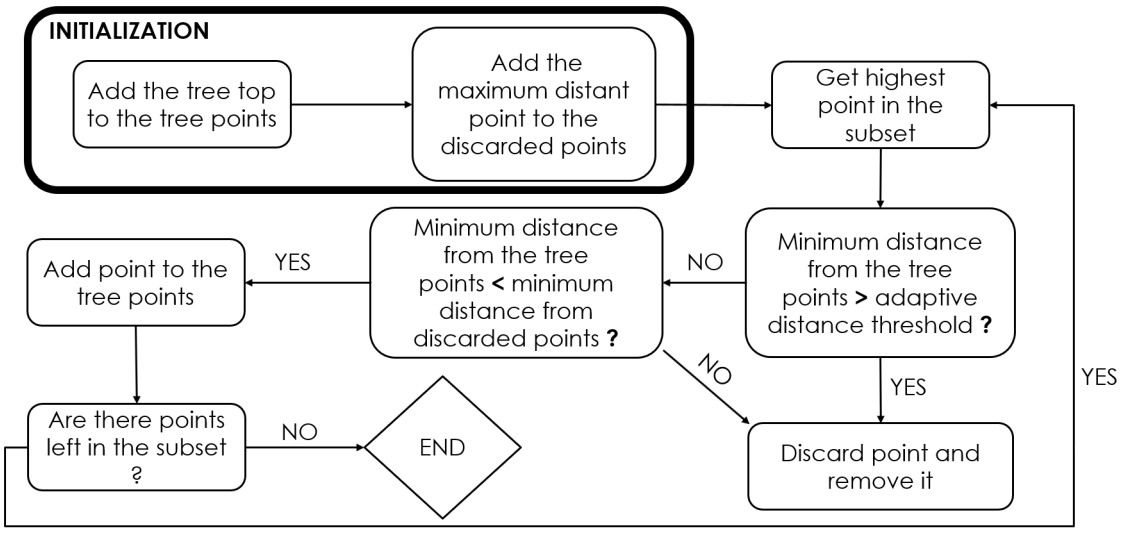

Figure 6.1: Flow chart summarizing the set of operations performed to identify a specific tree in the subset of points.

in a subset of the trees used as validation set followed by a human visual comparison of the produced results between the trees detected and the identified vegetation in the point cloud showed that the best fixed value for the distance threshold is 2.3 meters for the Kenya data.

In 6.2 we report a line plot with the number of founds trees among the 19 in the validation set of which we have the ground truth on the y axis and the distance threshold value used for the experiments on the x axis.

As shown in the plot, using a distance threshold value of 2.3 meters detected the highest number of trees with ground truth. In addition, this threshold value is also the one producing the best result with respect to the visual comparison performed among the original point cloud and the segmented trees.

6.2 Pseudo-code

Now we introduce the nomenclature used in the pseudo-code of the segmentation algorithm and the psuedo-code itself in order give a sequential ordering of the steps performed:

- **BS**: the radius of the ball where to search for neighbors in order to obtain the subset relative to a specific tree.
- **CT**: threshold on the number of points necessary to form a cluster. If the identified tree has less than CP points, then the cluster is discarded, given that the points in it are considered as noisy points generated by the ground/non

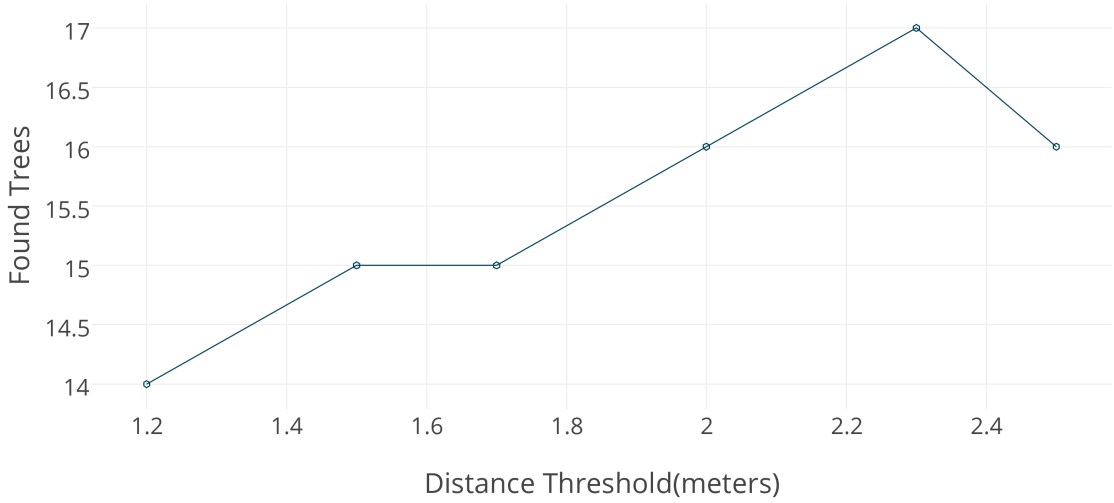

Figure 6.2: Line plot of the distance threshold values experimented (on the x axis) and of the number of trees with ground truth detected among the 19 in the validation set (on the y axis).

ground classification. By means of an empirical analysis we set this threshold to 30, given that we checked that no trees with less than 30 points were present in the point cloud.

- **HT**: height threshold used to determine the value of the adaptive distance threshold. This threshold was set to the 90% of the maximum height of points in the cloud after having analyzed the variation in trees' height in the areas.
- MD_C: the minimum distance of a given point from all the points in the cluster during the detection of a specific tree.
- MD_{NC}: the minimum distance of a given point from all the points discarded during the detection of a specific tree.
- **DT**: initial value chosen for the adaptive distance threshold.
- **NGP**: non ground points (points corresponding to vegetation).
- **CP**: list containing the points considered as part of the current tree.
- **NCP**: list containing the points discarded during the detection of a specific tree.
- **HP**: Highest Point.

- **TT**: Tree Top.

Before presenting the full pseudo-code of the algorithm, we first introduce the auxiliary functions called by the algorithm:

- **max_height_point(NPG)**: returns the point with maximum height among all the points in NGP.
- **get_points(TT, NGP, BS)**: this function builds a KDT with the NGP using the python implementation available in the library *sklearn*[1]. It then returns all the neighbors points within distance BS from the point TT in the created KDT.
- **compute_distance(point_a, point_b, n_dimensions)**: computes the euclidian distance between point *a* and point *b* in a number of dimensions equal to *n_dimensions*, which must be smaller or equal than the number of coordinates of the points.
- **get_most_distant_point(point, points)**: returns the point in the list of points passed to the function with maximum distance from the one given in input to the function using 3D euclidian distance.
- **compute_minimum_distance(point, points)** returns the minimum distance of the given point from all the points in the set passed to the function using 2D euclidian distance.
- **determine_threshold(HP, HT, DT)**: computes the adaptive distance threshold by returning DT if the height of HP is greater than HT, otherwise it returns $DT - 0.5$ meters, given that we could have some trees which are partially overlapping or which are lower than other trees to which they are close.

In Figure 6.3 we report a schematic representation of the different spacing between trees at different heights. As we can easily notice in Figure 6.3, it is clear that the spacing between the points of tree #2 and tree #1 is lower at a certain height, due to the fact that tree #2 has a lower height than our target tree and thus it makes sense to lower the distance threshold of a certain delta meters when the height of the candidate point is below the height threshold. The full pseudo-code of the segmentation algorithm is reported in Algorithm 3.

6.3 Segmentation Results

In this section we show the experiments performed and the results obtained using the segmentation algorithm on the point cloud data relative to the four areas where trees with ground truth are located. In the results obtained, each tree identified

Algorithm 3 Segmentation Algorithm

```
1:  procedure TreeSegmentation
2:      NGP ← filter the point cloud to keep only points with classification ≠ 2
3:      DT ← 2.3 meters
4:      CP ← ∅
5:      NCP ← ∅
6:      while size(NGP) ≠ 0 do
7:          TT ← max_height_point(NGP)
8:          CP.add(TreeTop)
9:          NGP.remove(TT)
10:         Subset ← get_points(TT, NGP, BS)
11:         NCP.add(get_most_distant_point(TT, Subset)
12:         while size(Subset) ≠ 0 do
13:             HP ← max_height_point(Subset)
14:             MD_C ← compute_minimum_distance_cluster(HP, CP)
15:             MD_NC ← compute_minimum_distance_not_cluster(HP, NCP)
16:             if MD_C > determine_threshold(HP, HT, DT) then
17:                 NCP.add(HP)
18:             else if MD_C < MD_NC then
19:                 CP.add(HP)
20:             else
21:                 NCP.add(HP)
22:             Subset.remove(HP)
23:         if size(CP) > CT then
24:             Save CP as an identified tree
25:         NGP.remove(CP)
26:         CP ← ∅
27:         NCP ← ∅
```

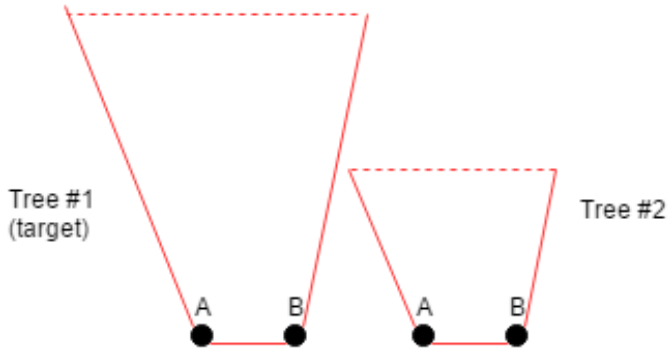

Figure 6.3: Example of the different spacing between trees at different heights.

has been marked with a different color in order to differentiate its points from the others.

The segmentation result for Area 1 is reported in figs. 6.4 and 6.5, where the algorithm correctly detected 11/12 trees of which we had the ground truth. By analyzing the original point cloud and the one obtained after the ground/non-ground separation we discovered that the missing tree (highlighted by a red ellipse) was classified as ground during the ground/non-ground separation phase, probably due to its low height.

The segmentation result for Area 2 is reported in figs. 6.6 and 6.7, where the algorithm identified 18/20 trees. By performing the same analysis on the point cloud relative to area 2 before and after the separation we discovered that the two missing trees (highlighted by red ellipses) were not detected because the points corresponding to their structure were classified as ground during the previous step of the framework.

After having encountered the same problem in Area 1 and 2, it is reasonable assuming that the separation algorithm has some issues in correctly classifying as vegetation the trees which have too low height. A positive aspect highlighted by the results obtained in area 2 is that the segmentation algorithm is able to correctly discern between overlapping trees, as shown by the partial overlap of different colors which then proceed highlighting well delineated trees structures.

The segmentation result for Area 3 is reported in figs. 6.8 and 6.9, where the algorithm detected 28/32 trees of which we had the ground truth. Many missing ones were not detected due to the same issue of the segmentation algorithm (trees highlighted by the red ellipses with ids 59, 73, 75), which we already discussed with respect to the segmentation results in area 2. For what concerns the tree highlighted by the red ellipse with id 99 instead, it was not detected due to the fact that it was too close to another bigger tree, thus the algorithm classified it as part of the bigger adjacent tree.

[1]implementation details at http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html

Figure 6.4: Area 1 original point cloud (trees with ground truth are highlighted by red ellipses).

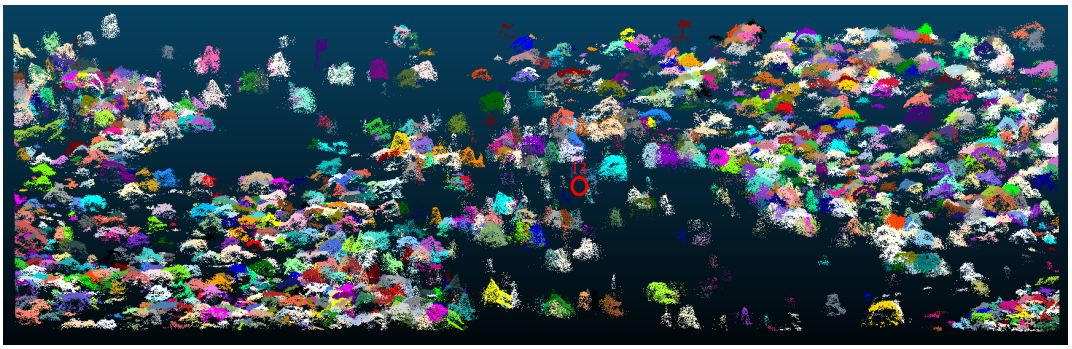

Figure 6.5: Trees identified by the segmentation algorithm in area 1 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

The segmentation result for Area 4 is reported in figs. 6.10 and 6.11, where the algorithm identified 48/55 trees. The majority of the missing trees (highlighted by the red ellipses with ids 179, 202, 203, 211) were classified as ground due to their low height. The fact that the segmentation issue related to low trees affects more Area 4 is predictable, given that Area 4 is the area with the highest percentage of bushes among the areas of interest. The remaining missing trees (highlighted by the red ellipses with ids 170, 171, 219) instead were not detected because they were too close to a bigger tree, thus being classified as part of the bigger one. Correctly separating trees which overlap too much or which are too close to each other is a hard task also for the human eye, thus it is reasonable that the algorithm commits some mistakes in the hardest cases.

In order to perform a more detailed analysis of the results we chose in each area a subarea in which it was possible to discern through a visual approach the various trees using *CloudCompare*, thus we were able to compare the trees segmented by the algorithm with the ones in the original point cloud. More in details, we estimated the TP by counting the trees which were correctly detected and the FP by counting the trees which were detected as individual ones but in the original point cloud they were part of bigger tree. The precision of the algorithm with respect to the trees

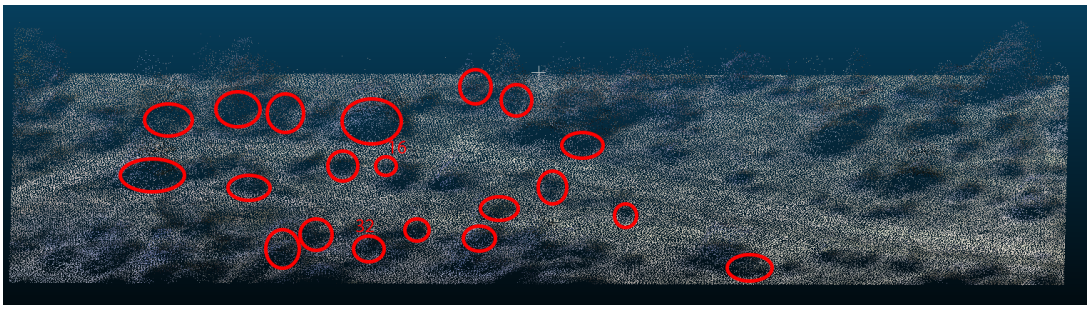

Figure 6.6: Area 2 original point cloud (trees with ground truth are highlighted by red ellipses).

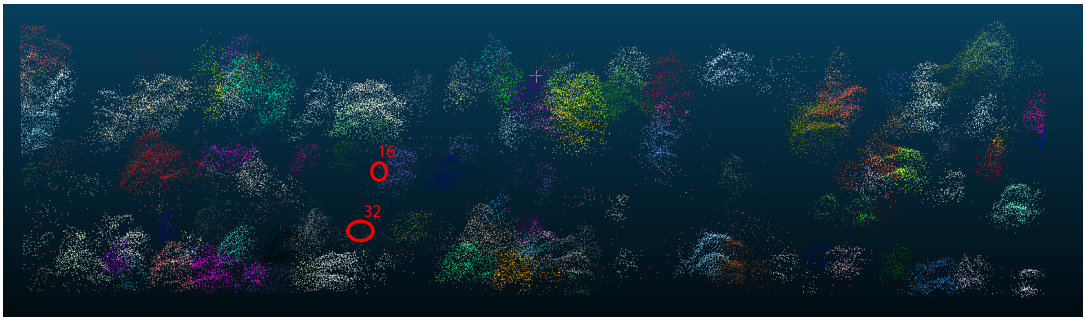

Figure 6.7: Trees identified by the segmentation algorithm in area 2 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

correctly identified in the four subareas is reported in Table 6.1.

Table 6.1: Results of the visual validation process over the four different sub areas chosen

Subarea	TP	FN	Precision
Subarea 1	43	12	78%
Subarea 2	36	7	83%
Subarea 3	30	8	79%
Subarea 4	55	13	80%

By analyzing the results reported in Table 6.1, we can notice that the segmentation algorithm correctly detected a very high percentage of the trees located in the subareas chosen. However, it must be said that this validation approach is not enough detailed to properly validate the results, but given that we don't have enough information about the true number of trees in the areas and their ground truth, due to the fact that the ones with ground truth are sparsed across the areas, we cannot perform a more detailed validation procedure. This does not represent a factor influencing in a significant way our final goal however, given that even if a single tree is misdetected, as more than one tree or as part of a bigger tree, we are interested

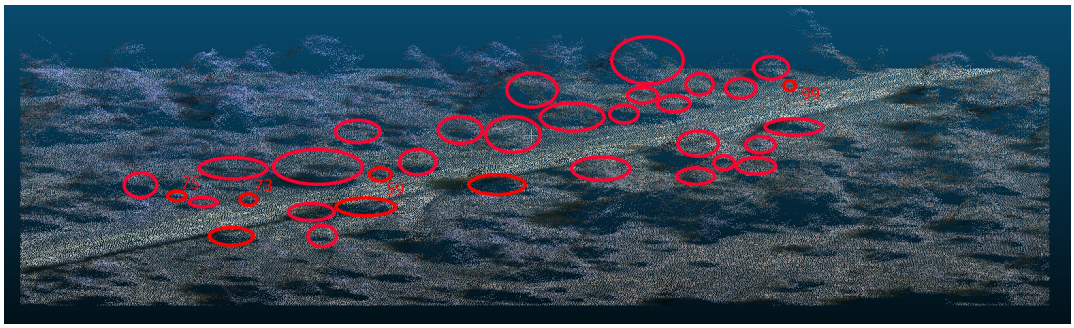

Figure 6.8: Area 3 original point cloud (trees with ground truth are highlighted by red ellipses).

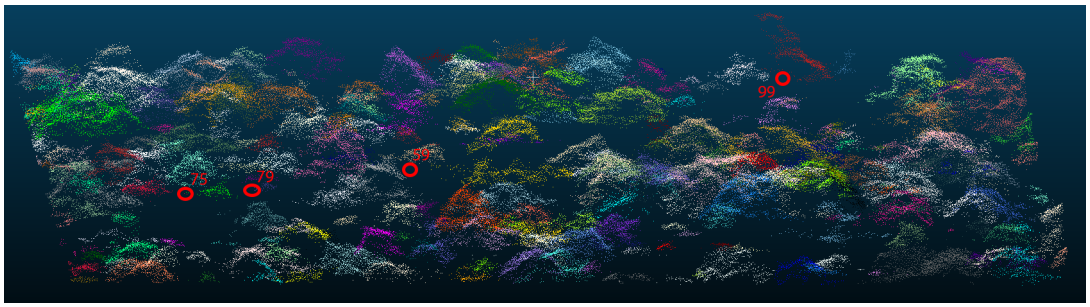

Figure 6.9: Trees identified by the segmentation algorithm in area 3 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

in inferring if it is going to represent a potential obstruction factor in the environment and this could be inferred correctly in both cases. About the trees classified as ground, this issue cannot be solved by the segmentation algorithm, considering that those trees are lost during the ground/non-ground separation phase.

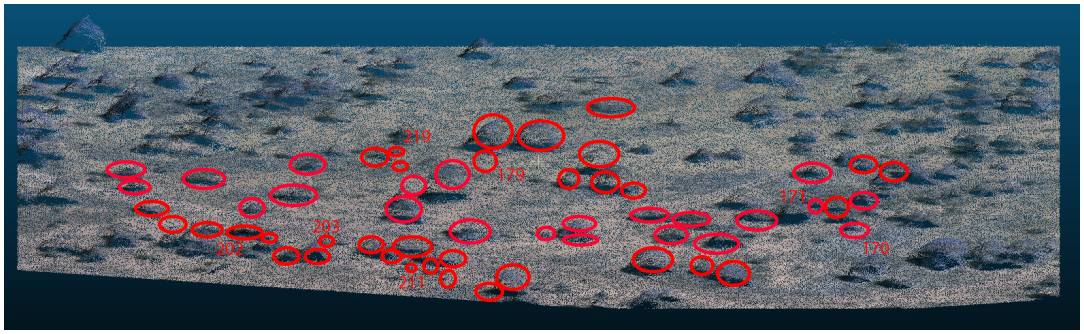

Figure 6.10: Area 4 original point cloud (trees with ground truth are highlighted by red ellipses).

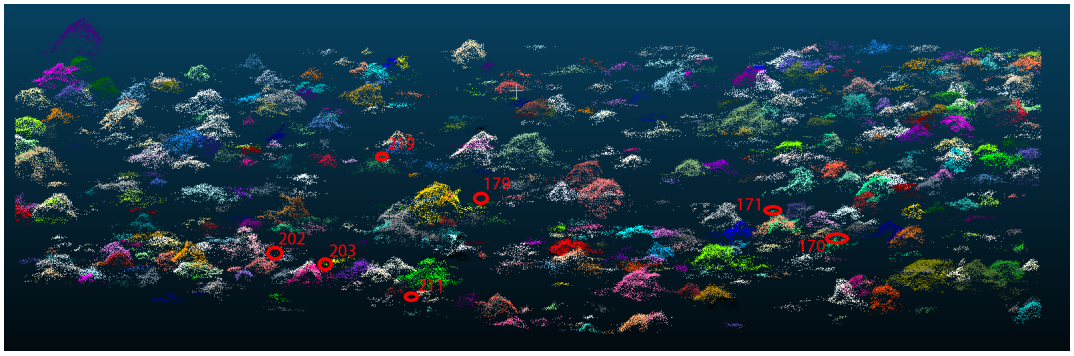

Figure 6.11: Trees identified by the segmentation algorithm in area 4 (missclassified trees among the ones with ground truth are highlighted by red ellipses).

Chapter 7

Visibility Features Extraction

After having obtained the individual vegetation elements among the whole vegetation points, we have to extract features embedding useful information from the trees point structure. In this chapter we will first go through the two main approaches chosen to extract features from the trees point structure and then we will analyze and compare the result obtained by performing inference with the extracted features. We followed two different approached for the features extraction procedure:

- In the first approach we engineered a certain number of chosen features considered suitable for the task at hand and extracted them from the trees' points structure. We will refer to this approach as a *manual extraction* of features, given that we chose before which features we want to extract from the data and then performed the extraction.
- In the second one we used the *stacked denoising autoencoders* neural network to automatically extract features from the trees points structure.

Before entering into details of the features extraction process, we first provide a detailed description of the inference task that must be performed.

7.1 Addressed Problem

Given the individually segmented trees through their points structure, our aim is extracting useful features that could be used to infer the visual obstruction potential of a specific tree.

In this setting we are interested in two classes: the "see" class marked as 1 (positive), corresponding to when the tree does not have visual obstruction potential, and the "notSee" class marked as 0 (negative), corresponding to the opposite case. For what concerns the mapping between the ground truth and the individual trees we exploited the fact that the UTM locations of the trees in the photos were known, thus they were easily mapped to the UTM locations of groups of points corresponding to specific trees.

7.2 Manual Features Extraction

In this section we describe in details each one of the features we considered suitable for the inference task at hand and the reasons behind their choice:

- **Height**: height is simply computed by taking the height coordinate of the point with maximum height in the cluster corresponding to the tree. The height is considered a useful feature given that the highest a tree is, the most probable is that it is going to be a potential source of obstruction.
- **Trunk Width**: the trunk width is computed by taking all the points at minimum height and computing the maximum distance between them. This feature is quite significant w.r.t the task, given the fact that the greater the trunk width is, the harder it will be to see through the tree.
- **Density**: density of the tree points is computed applying a *Gaussian Density Kernel* on all the points and then taking the mean of densities of all points in order to obtain one density value for the tree. Density is a crucial aspect with respect to our task, given that the more points corresponding to vegetation are dense, the harder it will be to see through the tree.
- **Delta X**: represents the maximum variation in the x coordinate of the points corresponding to the tree. The reason why we chose to compute this feature is because including the coordinates interval as features could help the classifier better understand which are the relevant coordinates intervals with respect to to the inference task.
- **Delta Y**: represents the maximum variation in the y coordinate of the points corresponding to the tree. The reason why we included it is the same for the *Delta X* variation.
- **Average Height**: it is simply computed as the mean of the height of all the points corresponding to the tree. This feature could bring useful insights to estimate how the height of each point influences the outcome.
- **Red Intensity**: the red intensity is computed as the variance of the red values of all points corresponding to the tree. Adding such feature could be useful in order to get the intensity of the tree color, which could be related to visibility issues among animals.
- **Green Intensity**: the green intensity is computed as the variance of the green value of all points corresponding to the tree. This feature could be useful for the same reason of the *Red Intensity* feature.
- **Blue Intensity**: the blue intensity is computed as the variance of the blue value of all points corresponding to the tree. This feature could be useful for the same reason of the *Red Intensity* feature.

7.3 Automatic Features Extraction

In order to automatically extract features from the trees points structure we used a *stacked denoised autoencoder* ANN, where we had to make some modifications to the input in order to make it fit to the ANN requirements:

- Given that neural networks expect a fixed size for the input but each tree is made of a variable number of points, we first found the *minimum size* tree and then sampled without replacement trees with size greater than the minimum one for a number of times equal to the minimum size found before, thus obtaining a fixed number of points for each tree.

- In order to make each input given to the neural network correspond to one tree, we concatenated the points picked from the tree structure into a unique array of size
$minimum\_size \cdot features\_number\_of\_point$.

- All the train and test data were normalized between 0 and 1 by means of a *min-max normalization*, given that neural networks are able to learn much more information from normalized data than not normalized ones.

Once prepared the input, we modified the code[1] already provided in the deep learning tutorials[2] based on the *Theano*[3] library.

The following modifications were applied to the original *stacked denoising autonecoders* source code:

- We modified the network in order to make it return the output of the last hidden layer instead of the output layer, given that it is where the most advanced features are extracted.

- The fine-tuning part was removed because we were not interested in using the neural network to perform a full inference task but just to extract features in an unsupervised fashion.

After the completion of all the preliminary steps, the network was trained with all the trees available in the four areas except the ones for which we had the ground truth, given that we used them as test set. Once the network was trained and tested, we used the obtained features to perform the inference task using different machine learning inference techniques in order to find the ones performing best for this specific task.

[1]The original code can be found at http://deeplearning.net/tutorial/code/SdA.py
[2]The tutorials can be found at http://deeplearning.net/tutorial/
[3]More information at http://deeplearning.net/software/theano/

7.4 Experiments and Results

Before performing the experiments, we first had to stratify our dataset, given that there was a significant dominance of the 'notSee' class. The reason why data should be stratified is that otherwise the classifier could be biased towards inferring the class appearing the most (majority class). To perform stratification, we checked which was the majority class and then we took all the instances corresponding to the minority class and an equal amount of instances from the majority class.

Once the dataset was balanced, we performed various experiments, which we are going to describe in the following sections together with the analysis of the result obtained. In order to take into account the fact that many machine learning methods produce different results due to the randomly chosen elements used for the initialization or to some functions used having multiple solutions, we trained and tested each method several times on our dataset. The final results are reported as the mean value plus-minus the standard deviation of the predictions obtained during the various calls. For the results relative to the deterministic machine learning methods which are insensitive to initialization or to the multiple solutions case, the standard deviations are zero, given that they always give the same results.

7.4.1 Inferring With Manually Extracted Features

The first experiment performed was trying to apply different machine learning techniques once we manually extracted the features presented in Section 7.2.

In order to properly evaluate the results we performed a LOOCV of which we report the results in Table 7.1.

Table 7.1: Accuracy, recall and precision results of the inference task with manually extracted features

Method	Accuracy	Recall	Precision
Random Forest	0.623 ± 0.017	0.463 ± 0.018	0.636 ± 0.023
Decision Tree	0.617 ± 0.023	0.542 ± 0.019	0.607 ± 0.022
Logistic Regression	0.703 ± 0.000	0.378 ± 0.000	0.809 ± 0.000
K-Nearest Neighbors	0.641 ± 0.000	0.512 ± 0.000	0.636 ± 0.000
Naive Bayes	0.609 ± 0.000	0.641 ± 0.000	0.581 ± 0.000
SVM Polynomial Kernel	0.500 ± 0.000	1.000 ± 0.000	0.500 ± 0.000

By analyzing the results in Table 7.1 we notice that all the tested classifiers performed better than the baseline, which is 0.5 given that we have stratified the

dataset.

More in details, the best results for accuracy and precision are 70% and 80%, obtained using respectively Decision Tree and Logistic Regression, while for recall the best result is obtained when using Naive Bayes with 61%. The worst result instead is obtained when using SVM with a polynomial kernel, which inferred exactly as the majority class.

The results obtained are all statistically significant according to an ANOVA test performed, which yielded a p-value of $1.65 \cdot e^{-14}$.

7.4.2 RGB Features Ablation

The next experiment performed was removing the RGB related features from the ones considered in order to check if extracting features relative to the RGB value was actually bringing additional information or not.

In order to validate the results obtained we performed again a LOOCV, obtaining the results reported in Table 7.2.

Table 7.2: Accuracy, recall and precision results of the inference task with manually extracted features except the rgb related features

Method	Accuracy	Recall	Precision
Random Forest	0.592 ± 0.025	0.449 ± 0.016	0.606 ± 0.031
Decision Tree	0.621 ± 0.034	0.523 ± 0.015	0.616 ± 0.029
Logistic Regression	0.578 ± 0.000	0.459 ± 0.000	0.586 ± 0.000
K-Nearest Neighbors	0.656 ± 0.000	0.452 ± 0.000	0.678 ± 0.000
Naive Bayes	0.625 ± 0.000	0.475 ± 0.000	0.633 ± 0.000
SVM Polynomial Kernel	0.547 ± 0.000	0.514 ± 0.000	0.545 ± 0.000

From Table 7.2 we can immediately notice that removing the RGB related features causes a drop in the validation metrics computed, meaning that with the RGB features we are extracting more valuable information for the task at hand.

We can notice that many of the tested classifiers perform worse or in the same way with respect to the previous experiment, exception made for Naive Bayes and SVM with polynomial kernel, which performs considerably better but still worse than the results obtained using also the RGB features in the previous experiment. In addition, results are not statistically significant according to an ANOVA test with p-value 0.967.

7.4.3 RGB Contribution Estimation

By analyzing the results of the previous experiment we noticed that the RGB features made a considerable contribution to the information available for the inference process. In order to determine the importance of this contribution among all the features, we decided to perform inference using only the information given by the RGB features. The LOOCV results of this experiment are reported in Table 7.3.

Table 7.3: Accuracy, recall and precision results of the inference task using the RGB features only.

Method	Accuracy	Recall	Precision
Random Forest	0.662 ± 0.014	0.476 ± 0.011	0.674 ± 0.017
Decision Tree	0.523 ± 0.022	0.528 ± 0.025	0.522 ± 0.021
Logistic Regression	0.703 ± 0.000	0.378 ± 0.000	0.809 ± 0.000
K-Nearest Neighbors	0.640 ± 0.000	0.512 ± 0.000	0.636 ± 0.000
Naive Bayes	0.609 ± 0.000	0.641 ± 0.000	0.581 ± 0.000
SVM Polynomial Kernel	0.500 ± 0.000	1.000 ± 0.000	0.500 ± 0.000

From the results shown in 7.3 we can notice that not only results are comparable with the ones obtained using all manually extracted features, but there is also a slight improvement in accuracy for Random Forest and recall for Decision Tree. The improvement obtained is statistically significant according to an ANOVA test with p-value $2.16 \cdot e^{-14}$. This last experiment showed that the RGB features are the features bringing the most valuable information to the inference task, outperforming all the other features.

7.4.4 Inferring With Automatically Extracted Features

The last experiment performed was inferring using the features automatically extracted with the stacked denoising autoencoders neural network. The main problem using neural networks is finding the best network configuration for the task. The only way to achieve this goal is testing different configurations and analyzing the drops or improvements in the results obtained and acting consequently. Generally speaking, one way to obtain an improvement is incrementing gradually the number of hidden layers and the neurons in each layer, based on the fact that a gradual deeper elaboration of the information at each layer could produce more valuable insights in output. By adopting this approach the improvement will be null or insignificant at a certain point, especially with respect to the trade-off between results

and the time needed in order to properly train the network. We again performed a LOOCV in order to properly evaluate the results obtained using each different configuration.

The first configuration tried consists in using all the trees in the four areas as input which do not correspond to one of the photos and in giving them in input to a stacked denoising autoencoder neural network with 3 hidden layers, the first with 256 neurons, the second with 512 neurons and a target number of features to be extracted equal to 1000 (corresponding to the number of neurons in the last hidden layer of the neural network). In Table 7.4 we report the results obtained with the first chosen configuration.

Table 7.4: Accuracy, recall and precision results of the inference task with automatically extracted features using 3 hidden layers

Method	Accuracy	Recall	Precision
Random Forest	0.753 ± 0.014	0.487 ± 0.007	0.764 ± 0.019
Decision Tree	0.676 ± 0.028	0.531 ± 0.022	0.663 ± 0.032
Logistic Regression	0.750 ± 0.000	0.333 ± 0.000	1.000 ± 0.000
K-Nearest Neighbors	0.734 ± 0.000	0.404 ± 0.000	0.826 ± 0.000
Naive Bayes	0.750 ± 0.000	0.354 ± 0.000	0.944 ± 0.000
SVM Linear Kernel	0.685 ± 0.000	0.363 ± 0.000	0.800 ± 0.000

As we can easily notice, Table 7.4 reports an overall significant improvement with respect to the results relative to the manually extracted features according to an ANOVA test with p-value 0.0021. In the best case we obtain 75% accuracy using Logistic Regression, Naive Bayes and Random Forest against the 68% relative to the manually extracted features case, 48% recall using Naive Bayes and 100% precision using Logistic Regression.

The result of this first experiment proves that using neural networks to extract deeper features is indeed promising and worth experimenting more in order to improve the results.

The second configuration tried consists in using again all the trees in the four areas which do not correspond to one of the photos as input for the stacked denoising autoencoder neural network with 4 hidden layers, the first with 256 neurons, the second with 512 neurons, the third with 1024 neurons and a target number of features to be extracted equal to 2000.

The result of the inference task with the second configuration adopted are reported in Table 7.5, where we can notice that performances are almost the same for Random Forest, Decision Tree, Logistic Regression and SVM with linear kernel with respect to the previous configuration, while we notice a significant improvement for

Table 7.5: Accuracy, recall and precision results of the inference task with automatically extracted features using 4 hidden layers

Method	Accuracy	Recall	Precision
Random Forest	0.750 ± 0.015	0.462 ± 0.006	0.784 ± 0.019
Decision Tree	0.665 ± 0.030	0.512 ± 0.018	0.662 ± 0.032
Logistic Regression	0.750 ± 0.000	0.333 ± 0.000	1.000 ± 0.000
K-Nearest Neighbors	0.750 ± 0.000	0.416 ± 0.000	0.833 ± 0.000
Naive Bayes	0.734 ± 0.000	0.361 ± 0.000	0.895 ± 0.000
SVM Linear Kernel	0.671 ± 0.000	0.372 ± 0.000	0.761 ± 0.000

KNN and Naive Bayes, according to an ANOVA test with p-value 0.0079.

The third configuration tried consists in using again all the trees in the four areas which do not correspond to one of the photos and in giving them in input to a stacked denoising autoencoder neural network with 5 hidden layers, the first with 256 neurons, the second with 512 neurons, the third with 1024 neurons, the fourth with 2048 neurons and a target number of features to be extracted equal to 3000. The results of the third configuration tried are reported in Table 7.6.

Table 7.6: Accuracy, recall and precision results of the inference task with automatically extracted features using 5 hidden layers

Method	Accuracy	Recall	Precision
Random Forest	0.775 ± 0.012	0.492 ± 0.008	0.782 ± 0.002
Decision Tree	0.753 ± 0.029	0.504 ± 0.017	0.750 ± 0.041
Logistic Regression	0.734 ± 0.000	0.240 ± 0.000	0.941 ± 0.000
K-Nearest Neighbors	0.828 ± 0.000	0.453 ± 0.000	0.889 ± 0.000
Naive Bayes	0.750 ± 0.000	0.354 ± 0.000	0.944 ± 0.000
SVM Linear Kernel	0.750 ± 0.000	0.333 ± 0.000	1.000 ± 0.000

The result shown in Table 7.6 confirms the theory that incrementing the numbers of layers and its neurons in a gradual way leads to a significative improvement, confirmed by an ANOVA test with p-value of 0.0026 for this last attempt.

More in details, the above table reports an improvement in accuracy and precision respectively with a 83% obtained with KNN and a 100% obtained with Logistic Regression, while we observe a slight drop in the recall with an highest result of 48%

obtained by Random Forest against the 51% obtained by Decision Tree when using the 4 hidden layers network configuration.

Even if we managed to obtain a significant improvement in the validation metrics used with these last attempts, when using neural networks a trade off between training time and results obtained must be always taken into account, especially when incrementing the number of hidden layers used and the neurons in them.
Experiments with more hidden layers than in the last configuration presented yielded results which did not bring any relevant improvement in any of the metrics used while the training time increased considerably. From the results of the experiments with more hidden layers we concluded that incrementing the number of hidden layers and their neurons was not improving performances anymore. An explanation to this limitation could be that not enough trees are given in input to the neural network, thus maybe increasing too much the number of hidden layers with too few trees in input does not allow to extract enough information at too deep levels of the network, thus leaving the result unchanged. For this reason we decided to avoid testing configuration with even more hidden layers, given the fact that we already used all the available trees as input.

7.4.5 Results Comparison

In order to make a comparison among the results obtained in the different experiments and configurations used, in Figure 7.1 we report a bar chart with the highest results obtained in each performed experiment.

As reported in Figure 7.1, the best results for accuracy is obtained by the 5HL neural network with KNN, while for recall with the 3HL one by using Random Forest, excluding the 100% obtained when SVM was behaving exactly like the majority class, and for precision with all the neural network configurations by using Logistic Regression and SVM.

This last comparison highlights that the best results with respect to the evaluation metrics used were obtained by means of neural networks, confirming the deep learning effectiveness for the task at hand. From the bar graph we can also notice that overall we obtained low results for recall with a highest value of 60%, meaning that the inference algorithms used are biased toward the negative class with respect to the dataset used, considering that we have an high number of FN which lower the recall percentage.

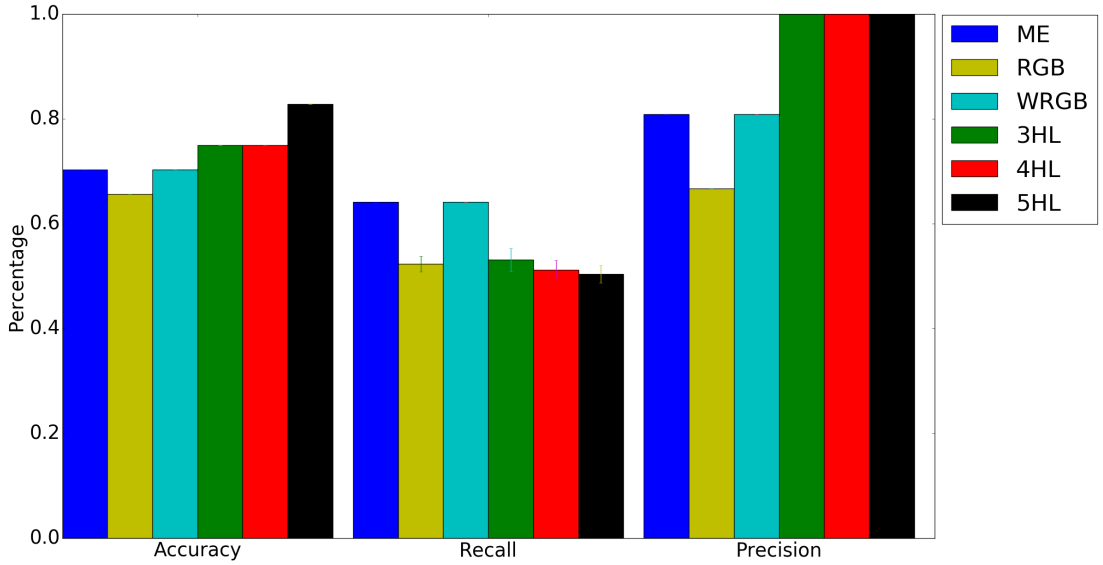

Figure 7.1: Highest results obtained for each metric during the various experiments where error bars are reported only for results with standard deviation different from zero. The legend is the following:

- ME: *manually extracted features*
- WRGB: *RGB features ablation test*
- RGB: *RGB features only*
- nHL: *configuration with **n** hidden layers*

Chapter 8

Visibility Networks

Once we obtained the visual obstruction potential labels, one of the most immediate uses for the information extracted is building a visibility network among individuals located in the environment. The idea is computing a different visibility network at each timestamp in which each individual in the environment has a specific location. This chapter is focused on describing the intuitive procedure used to efficiently analyze the lines of sight among individuals, given that edges of the visibility network will be drawn according to the results obtained with this procedure. The reason why we chose this specific example to showcase a possible use of the information extracted is that the approach used to build a visibility network among animals in forested environments could be generalized to generic individuals in any kind of environment, given the availability of the necessary information.

The data we used to build the visibility network are the baboons GPS trajectories. Of the whole trajectories we considered only subsets inside Area 2 geographic boundaries, given that in order to perform the analysis of the lines of sight we need the ground truth, which is fully available only for Area 2 thanks to the fact that we manually labeled each tree by analyzing the point cloud with *CloudCompare*. We are aware that labeling the trees by manually analyzing the point cloud is a very long shot but these information could be extracted in a real setting using the proposed framework. Once filtered the GPS trajectories according to Area 2 geographic boundaries, we have a total amount of 86505 timestamps during which the highest number of baboons were inside Area 2 geographic boundaries. The total amount of available timestamps for the baboons trajectories is 1462848, while timestamps in which baboons were simultaneously in Area 2 go from 92191 to 178696. Among timestamps relative to Area 2, we computed the visibility networks for timestamps from 178495 to 178595, with a total amount of 100 visibility networks.

In the following sections we will give a detailed description of the method used to analyze the lines of sight among individuals together with suitable pseudo-code, then we will show the results obtained in the experiments performed and the visibility networks obtained.

8.1 Lines of Sight Analysis

In order to build the visibility network for a specific timestamp we need an algorithm to analyze the line of sight among two individuals, which could be used to determine whether a specific edge in the network must be drawn or not. An explanatory image of analysis of the line of sight among two individuals in the environment is reported in Figure 8.1.

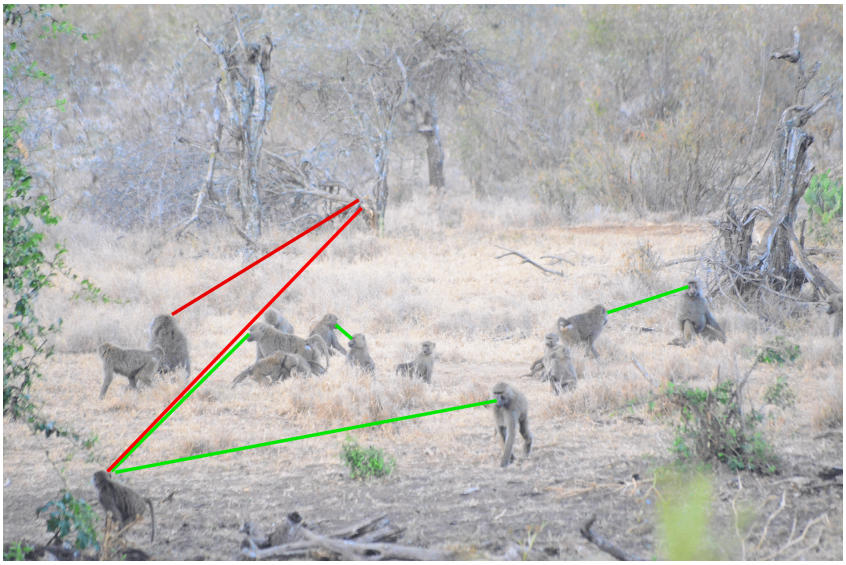

Figure 8.1: Baboons habitat with not obstructed lines of sight in green and obstructed lines of sight in red (baboons behind the tree pointed by the red lines cannot be seen).

In order to speed up the analysis and the whole processing operations we approximated the geometry of entities which must be modeled for the lines of sight analysis to a 2D world. More in details, the idea is approximating each tree with an ellipse built using only two of the three available coordinates. After having tested the results obtained with different sets of coordinates, we chose the Easting and Northing coordinate to build the ellipse, given the fact that they gave us the best approximation according to the results obtained. In order to model the geometric entities necessary to analyze the lines of sight we used the *python* library *"simpy"*[1].

Once geometrically modeled the entities, the following information is necessary to properly analyze the lines of sight:

- The position of the two individuals in the UTM system and their heights in meters.
- The variation on the Easting and Northing coordinates computed using the positions at successive timestamps in the GPS trajectories.

[1]The library can be found at http://docs.sympy.org/latest/index.html

The positions of the two individuals are used to identify which are the trees located between them while the variation of the UTM coordinates is used to perform an orientation analysis aimed at determining if the two individuals are oriented one towards the other. Based on the fact that if the first individual is able to see the second it is not guaranteed that the second is able to see the first, the line of sight analysis will check the visibility only under the first individual perspective. The reason behind this choice is that the line of sight analysis procedure is called for any possible couple in both possible orders, thus it will be checked if the second individual is able to see the first as well.

Once received all the necessary information in input, the following operations are performed to determine whether the first individuals is able to see the second one or not:

I) Perform an orientation analysis using the variation of the UTM coordinates. More in details, if the first individual is located further or before geographically with respect to the second and the variation of the UTM coordinates is such that the first individual is oriented in a way that does not allow him to see the second, then the procedure terminates otherwise proceed to step **II**.

II) Compute the 2D line passing between the two individuals using the Easting and Northing coordinate.

III) Collect the trees with obstruction potential using the inferred labels and keep only the ones in range of the Easting and Northing coordinates of the individuals.

IV) For each of the collected trees, the following checks are performed:

- **i**) Check if the height of the tree is greater than the one of both individuals. In case of positive outcome, proceed to step **(b)**, otherwise skip the tree given that it is too low in order to obstruct the line of sight.
- **ii**) Compute an ellipse approximating the tree shape using the Easting and Northing coordinate. This is done by using as center for the ellipse the centroid (average of all points for each coordinate) of the tree points for the chosen coordinates, as horizontal radius the value corresponding to half of the maximum variation in the Easting coordinate and as vertical radius the value corresponding to half of the maximum variation in the Northing coordinate.
- **iii**) Check if the line connecting the two individuals intersects the ellipse approximating the tree. In case of positive outcome, then the line of sight is considered obstructed and the the first individual is not able to see the other.

V) If no tree with visual obstruction potential crosses the line of sight among the two individuals then the line of sight is considered not obstructed and a directed edge is drawn in the network to represent that at the relative timestamp the first individual is able to see the second.

Once the procedure terminates, the method returns the network with the eventually added edges.

8.2 Pseudo-code

For completeness we provide also the pseudo-code of the algorithm used to analyze the line of sight among two individuals. Before showing the complete pesudocode, we first describe a set of auxiliary functions and parameters used in the algorithm:

- **a**, **b**: lists corresponding to the locations of the two individuals which in sequence contain the Easting coordinate, the Northing coordinate and the height of individuals.
- **dx**, **dy**: variation of the Easting and Northing coordinates of the first individual, computed as the difference between the location relative to the successive timestamp in the GPS trajectory and the location relative to the actual timestamp.
- **compute_line(a, b)**: this function uses the 'simpy' library to construct a 2D line passing between the two individual locations a and b using the Easting and Northing coordinates.
- **get_trees_with_obstruction_potential_in_range(a, b)**: this function returns the trees which were inferred to be visual obstruction potential elements and which are also in range of the Easting and Northing coordinates of the two individuals.
- **height(point)**: returns the height of the point.
- **compute_ellipse(center, h_radius, v_radius)**: construct an ellipse having as center the one passed to the function, as horizontal radius and vertical radius the ones specified in input.
- **check_intersection(line, ellipse)**: checks if the line in input intersects the ellipse given in input to the function.

The set of operations described in Section 8.1 corresponds to the pseudo-code reported in Algorithm 4.

Algorithm 4 Lines of Sight Analysis

```
1:  procedure CheckVisibility(a, b, dx, dy, network)
2:      if a[0] ≥ b[0] ∧ a[1] ≥ b[1] ∧ dx ≥ 0 ∧ dy ≥ 0   ∨
3:          a[0] ≤ b[0] ∧ a[1] ≤ b[1] ∧ dx ≤ 0 ∧ dy ≤ 0
4:          a[0] ≥ b[0] ∧ a[1] ≤ b[1] ∧ dx ≥ 0 ∧ dy ≤ 0
5:          a[0] ≤ b[0] ∧ a[1] ≥ b[1] ∧ dx ≤ 0 ∧ dy ≥ 0  then
6:          check ← False
7:      else
8:          check ← True
9:      if check then
10:         line ← compute_line(a, b)
11:         trees ← get_trees_with_obstruction_potential_in_range(a, b)
12:         for each tree in trees do
13:             if height(tree) < height(a) ∧ height(tree) < height(b) then
14:                 continue
15:             else
16:                 ellipse ← compute_ellipse(tree_centorid, delta_x/2, delta_y/2)
17:                 if check_intersection(line, ellipse) then
18:                     return network
19:         network.add_direct_edge(a, b)
20:     return network
```

8.3 Results and Experiments

In order to properly test the lines of sight analysis method proposed we used the labels obtained by manually analyzing the point cloud surface relative to Area 2. Once that we have the labels, we called the method a certain number of times on random locations in Area 2 range. The outcome was then compared with the result obtained by performing a manual analysis with *CloudCompare* of the point cloud with respect to the lines of sight corresponding to the random locations generated for the experiment. By performing 15 calls of the procedure with random locations we obtained a 86% of accuracy, analyzing correctly 13/15 lines of sight. The visual check performed to validate the results does not have an high reliability however, given that labeling in a proper way the trees just by looking at their points structure in the point cloud is a very long shot. Anyway, given the low availability of ground truth for the trees in the four areas, we did not have a different option to perform a more reliable validation procedure for the method proposed.

8.4 Visibility Network Construction

Once defined and tested a valid procedure to analyze the line of sight among two individuals in the environment, we called this procedure for any possible couple in both orders with respect to the baboons which were in Area 2 at the same timestamps, thus generating a visibility network for each timestamp depicting the visual contact among individuals. The directed graphs corresponding to the visibility networks computed for timestamps going from 178573 to 178578 are reported in figs. 8.2 to 8.7 where numbers in the nodes correspond to the ids of baboons.

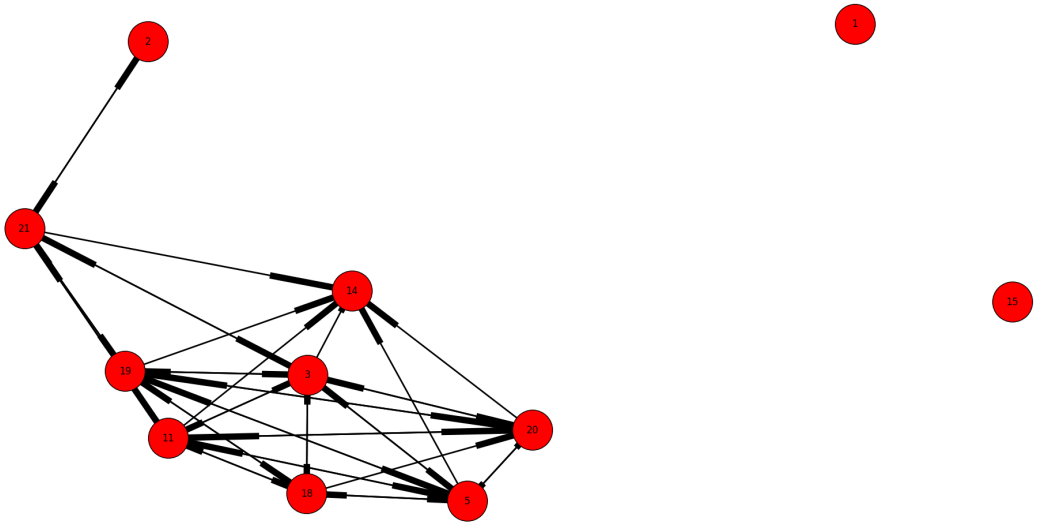

Figure 8.2: Visibility network at timestamp 178573.

As we can see in in figs. 8.2 to 8.7, the dynamic of the sighting among baboons is correctly captured, as remarked by the presence of many directed edges and their variability.

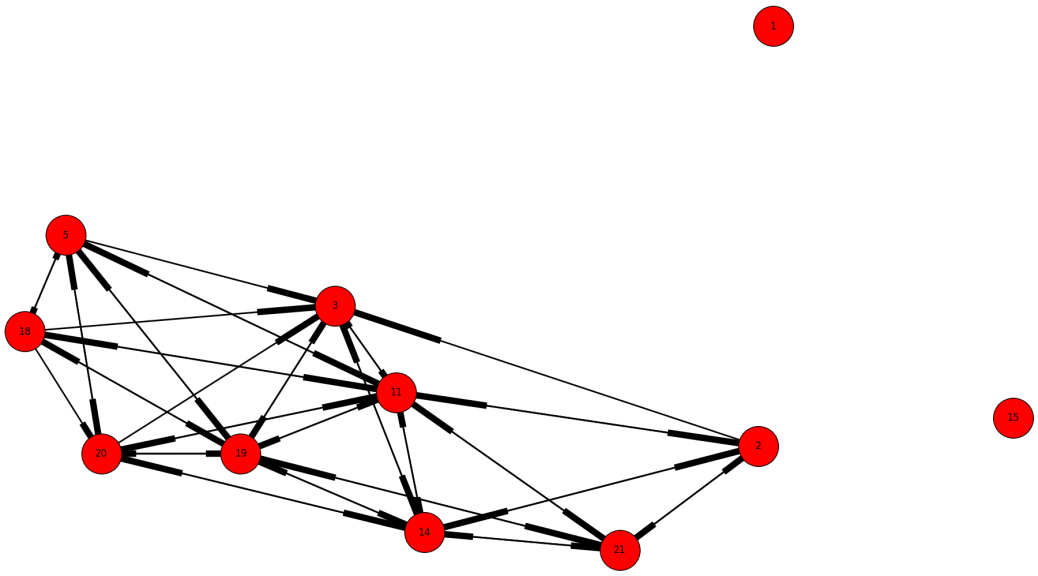

Figure 8.3: Visibility network at timestamp 178574.

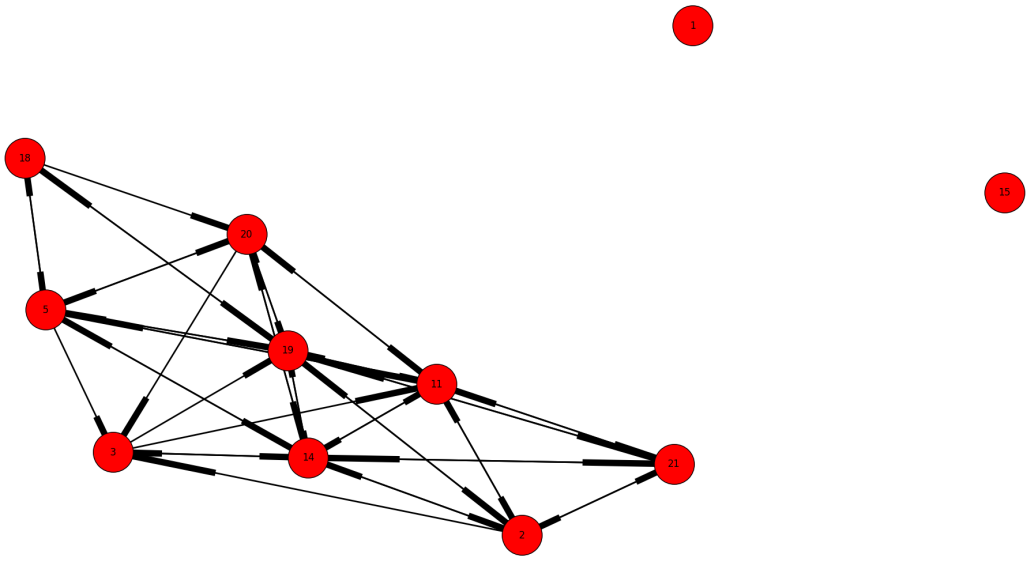

Figure 8.4: Visibility network at timestamp 178575.

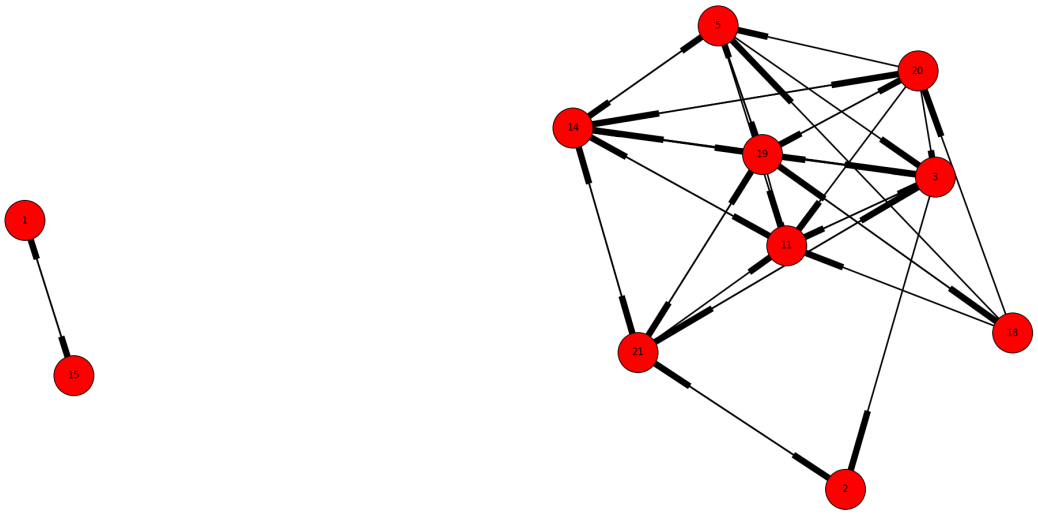

Figure 8.5: Visibility network at timestamp 178576.

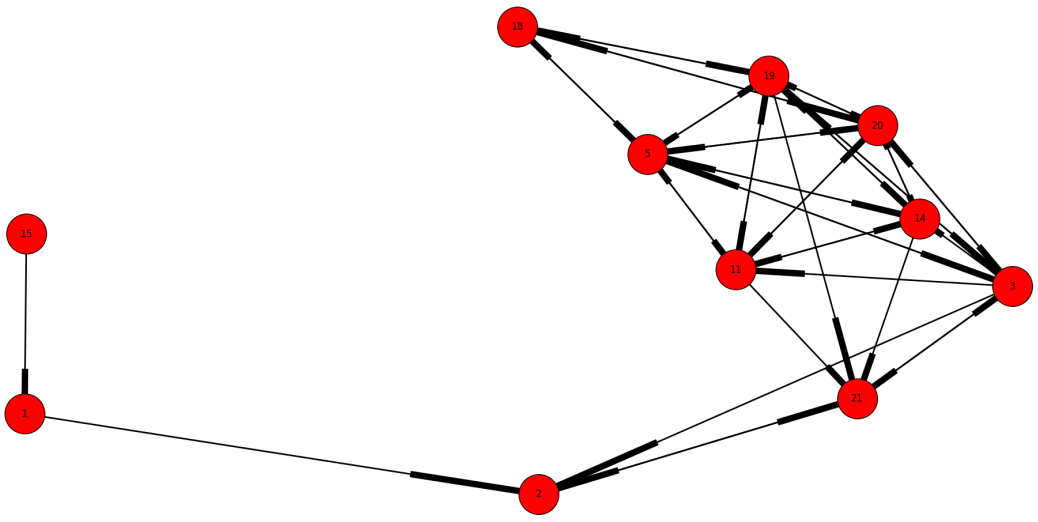

Figure 8.6: Visibility network at timestamp 178577.

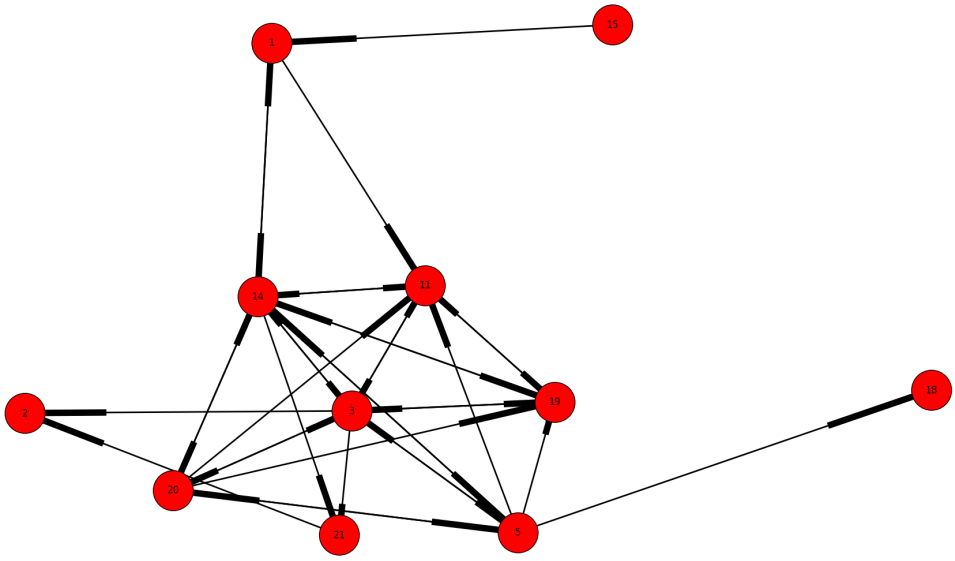

Figure 8.7: Visibility network at timestamp 178578.

Chapter 9

Conclusion

9.1 Final Considerations

In this thesis we proposed a framework capable of extracting information about terrain and vegetation starting from raw point cloud data corresponding to forested surfaces. From the vegetation data, each individual vegetation element is then identified by grouping together the corresponding points through a clustering approach. The terrain and vegetation data are then used to infer the visual obstruction potential of the vegetation elements identified by extracting features from their points structure and by performing inference using the extracted features. We provided also a practical application of the data extracted by building a visibility network for each timestamp leveraging on a lines of sight analysis procedure among individuals located in the environment and on the visual obstruction potential inferred starting from the point structure of the vegetation elements identified. The computation of visibility networks for each timestamp however is just one example of how the information extracted could be used, many other uses are feasible, such as supporting the creation of a fully immersive experience of the animals movements and actions in the environment, which would help understanding how animals perceive their habitat during their everyday life, thus providing really useful insights for studies related to animal behavior. To identify vegetation elements we described a fast and effective segmentation algorithm that could be used to segment trees in heterogeneously shaped forests, once properly tuned, thus providing information on vegetation that could be used to infer even more on the environment itself or on the elements linked to it. To conclude, even if the final objective of this thesis is a very domain specific one, single steps of this framework could be easily adapted and integrated in frameworks with different final objectives. For instance, with the ground truth of the specific task that one wishes to perform, the automatic unsupervised features extraction approach making use of the stacked denoising autoencoder neural network could be easily applied to obtain features for another domain specific inference task.

9.2 Future Work

There are also many feasible extension for the work done in this thesis. One of the feasible future improvement could be estimating the threshold value used during the tree segmentation phase in a dynamic way, directly from the characteristics of the points in the point cloud surface, thus making the procedure adaptable to any kind of forested environment without the time-consuming tuning procedure.

For what concerns the visual obstruction potential inference, training the neural network with an even higher number of inputs, if available, and using an higher number of hidden layers and neurons in the configurations adopted could be an interesting experiment. This experiment could be used to verify whether the limit to the deep learning improvement was just the limited number of training instances or was due to some other aspect of the training procedure, given that in this thesis our inputs was limited to the trees located in the four areas of interest.

Another improvement could be obtained in the analysis of the lines of sighs by modeling the relative forested environment by means of 3D geometry. With a 3D modeling of the environment, it would be possible to perform a more detailed analysis of the lines of sight among individuals, exploiting the orientation at which the lines of sight intersect the vegetation elements and modeling in a more detailed way the vegetation elements themselves without any kind of approximation.

Bibliography

[1] A Aizerman, Emmanuel M Braverman, and LI Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[2] Vitale B. Inferring high resolution terrain, vegetation, and lines of sight models from point cloud data. *University of Illinois at Chicago*, 2016.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[4] Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Ian C Briggs. Machine contouring using minimum curvature. *Geophysics*, 39(1):39–48, 1974.

[7] Y Chang, A Habib, D Lee, and J Yom. Automatic classification of lidar data into ground and non-ground points. *International archives of Photogrammetry and Remote Sensing*, 37:463–468, 2008.

[8] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *The Journal of Machine Learning Research*, 11:1471–1490, 2010.

[9] Qi Chen. Airborne lidar data processing and information extraction. *Photogrammetric engineering and remote sensing*, 73(2):109, 2007.

[10] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[11] Joseph F Dracup. *Geodetic Surveying 1940-1990*. US Department of Commerce, National Oceanic and Atmospheric Administration, National Ocean Service, 1995.

[12] M Elmqvist. Ground surface estimation from airborne laser scanner data using active shape models. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):114–118, 2002.

[13] JB Elsner, TH Jagger, and EA Fogarty. Visibility network of united states hurricanes. *Geophysical Research Letters*, 36(16), 2009.

[14] Jeffrey S Evans and Andrew T Hudak. A multiscale curvature algorithm for classifying discrete return lidar in forested environments. *Geoscience and Remote Sensing, IEEE Transactions on*, 45(4):1029–1038, 2007.

[15] Brian Sidney Everitt. The cambridge dictionary of statistics. *The Indian Journal of Statistics*, 69(4):887–888, 2007.

[16] Christian Früh and Avideh Zakhor. 3d model generation for cities using aerial photographs and ground level laser scans. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–31. IEEE, 2001.

[17] Qinghua Guo, Wenkai Li, Hong Yu, and Otto Alvarez. Effects of topographic variability and lidar sampling density on several dem interpolation methods. *Photogrammetric Engineering & Remote Sensing*, 76(6):701–712, 2010.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Unsupervised learning*. Springer, 2009.

[19] Johannes Heinzel and Barbara Koch. Exploring full-waveform lidar parameters for tree species classification. *International Journal of Applied Earth Observation and Geoinformation*, 13(1):152–160, 2011.

[20] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[21] Shridhar D Jawak, Satej N Panditrao, and Alvarinho J Luis. Validation of high-density airborne lidar-based feature extraction using very high resolution optical remote sensing data. *Advances in Remote Sensing*, 2013, 2013.

[22] I Korpela, B Dahlin, H Schäfer, E Bruun, F Haapaniemi, J Honkasalo, S Ilvesniemi, V Kuutti, M Linkosalmi, J Mustonen, et al. Single-tree forest inventory using lidar and aerial images for 3d treetop positioning, species recognition, height and crown width estimation. In *Proceedings of ISPRS workshop on laser scanning*, pages 227–233, 2007.

[23] Olivier Küng, Christoph Strecha, Antoine Beyeler, Jean-Christophe Zufferey, Dario Floreano, Pascal Fua, and François Gervaix. The accuracy of automatic photogrammetric techniques on ultra-light uav imagery. In *UAV-g 2011-Unmanned Aerial Vehicle in Geomatics*, number EPFL-CONF-168806, 2011.

[24] Heezin Lee, K Clint Slatton, BE Roth, and WP Cropper Jr. Adaptive clustering of airborne lidar data to segment individual tree crowns in managed pine forests. *International Journal of Remote Sensing*, 31(1):117–139, 2010.

[25] Qi Li and Jeffrey Scott Racine. *Nonparametric econometrics: theory and practice*. Princeton University Press, 2007.

[26] Wenkai Li, Qinghua Guo, Marek K Jakubowski, and Maggi Kelly. A new method for segmenting individual trees from the lidar point cloud. *Photogrammetric Engineering & Remote Sensing*, 78(1):75–84, 2012.

[27] An Lidar and Of plant canopies. Lidar remote sensing for ecosystem studies.

[28] Kevin Lim, Paul Treitz, Michael Wulder, Benoît St-Onge, and Martin Flood. Lidar remote sensing of forest structure. *Progress in physical geography*, 27(1):88–106, 2003.

[29] Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

[30] Cheng-Yuan Liou, Jau-Chi Huang, and Wen-Chie Yang. Modeling word perception using the elman network. *Neurocomputing*, 71(16):3150–3157, 2008.

[31] Jordi Llorens, Emilio Gil, Jordi Llop, and Meritxell Queraltó. Georeferenced lidar 3d vine plantation map generation. *Sensors*, 11(6):6237–6256, 2011.

[32] Yu Long. Visibility graph network analysis of gold price time series. *Physica A: Statistical Mechanics and its Applications*, 392(16):3374–3384, 2013.

[33] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[34] Helena Mäkelä and Anssi Pekkarinen. Estimation of forest stand volumes by landsat tm imagery and stand-level field-inventory data. *Forest ecology and management*, 196(2):245–255, 2004.

[35] Xuelian Meng, Nate Currit, and Kaiguang Zhao. Ground filtering algorithms for airborne lidar data: A review of critical issues. *Remote Sensing*, 2(3):833–860, 2010.

[36] Xuelian Meng, Le Wang, José Luis Silván-Cárdenas, and Nate Currit. A multi-directional ground filtering algorithm for airborne lidar. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(1):117–124, 2009.

[37] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[38] Felix Morsdorf, Erich Meier, Britta Allgöwer, and Daniel Nüesch. Clustering in airborne laser scanning raw data for segmentation of single trees. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34(part 3):W13, 2003.

[39] Bastiaan Notebaert, Gert Verstraeten, Gerard Govers, and Jean Poesen. Qualitative and quantitative applications of lidar imagery in fluvial geomorphology. *Earth Surface Processes and Landforms*, 34(2):217–231, 2009.

[40] Doyun Park, Tek-Jin Nam, and Chung-Kon Shi. Designing an immersive tour experience system for cultural tour sites. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1193–1198. ACM, 2006.

[41] Lawrence R Rabiner. *Multirate digital signal processing*. Prentice Hall PTR, 1996.

[42] Raul Rojas. The backpropagation algorithm. In *Neural networks*, pages 149–182. Springer, 1996.

[43] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[44] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[45] Martin Rutzinger, Bernhard Höfle, Markus Hollaus, and Norbert Pfeifer. Object-based point cloud analysis of full-waveform airborne laser scanning data for urban vegetation classification. *Sensors*, 8(8):4505–4528, 2008.

[46] Ricardo Sanz-Cortiella, Jordi Llorens-Calveras, Jaume Arnó-Satorra, Manel Ribes-Dasi, Joan Masip-Vilalta, Ferran Camp, Felip Gràcia-Aguilá, Francesc Solanelles-Batlle, Santiago Planas-DeMartí, Tomàs Pallejà-Cabré, et al. Innovative lidar 3d dynamic measurement system to estimate fruit-tree leaf area. *Sensors*, 11(6):5769–5791, 2011.

[47] Fayez Tarsha-Kurdi, Tania Landes, Pierre Grussenmeyer, et al. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In *Proceedings of the ISPRS Workshop on Laser Scanning*, volume 36, pages 407–412, 2007.

[48] Dirk Tiede, Gregor Hochleitner, and Thomas Blaschke. A full gis-based workflow for tree identification and tree crown delineation using laser scanning. In *ISPRS Workshop CMRT*, volume 5, page 2005, 2005.

[49] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[50] Philip D Wasserman. *Advanced methods in neural computing*. John Wiley & Sons, Inc., 1993.

[51] Aloysius Wehr and Uwe Lohr. Airborne laser scanning—an introduction and overview. *ISPRS Journal of Photogrammetry and Remote Sensing*, 54(2):68–82, 1999.