3D Scan Campaign Classification with Representative Training Scan Selection

by

Christopher Pocock

Dissertation presented for the degree of
Master of Science in the department of Computer Science

UNIVERSITY OF CAPE TOWN

October 2019

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the required convention for citation and referencing. Each contribution to and quotation in this thesis from the work(s) of other people has been attributed, and has been cited and referenced.
3. This thesis is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
5. I acknowledge that copying someone else's work, or part of it, is wrong, and declare that this is my own work.

Christopher Pocock, October 2019

Abstract

Point cloud classification has been shown to effectively classify points in 3D scans, and can accelerate manual tasks like the removal of unwanted points from cultural heritage scans. However, a classifier's performance depends on which classifier and feature set is used, and choosing these is difficult since previous approaches may not generalise to new domains. Furthermore, when choosing training scans for campaign-based classification, it is important to identify a descriptive set of scans that represent the rest of the campaign. However, this task is increasingly onerous for large and diverse campaigns, and randomly selecting scans does not guarantee a descriptive training set.

To address these challenges, a framework including three classifiers (Random Forest (RF), Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP)) and various point features and feature selection methods was developed. The framework also includes a proposed automatic representative scan selection method, which uses segmentation and clustering to identify *balanced*, *similar* or *distinct* training scans. The framework was evaluated on four labelled datasets, including two cultural heritage campaigns, to compare the speed and accuracy of the implemented classifiers and feature sets, and to determine if the proposed selection method identifies scans that yield a more accurate classifier than random selection.

It was found that the RF, paired with a complete multi-scale feature set including covariance, geometric and height-based features, consistently achieved the highest overall accuracy on the four datasets. However, the other classifiers and reduced sets of selected features achieved similar accuracy and, in some cases, greatly reduced training and prediction times. It was also found that the proposed training scan selection method can, on particularly diverse campaigns, yield a more accurate classifier than random selection. However, for homogeneous campaigns where variations to the training set have limited impact, the method is less applicable. Furthermore, it is dependent on segmentation and clustering output, which require campaign-specific parameter tuning and may be imprecise.

Acknowledgements

Thank you to my supervisor, Prof. Patrick Marais, for his guidance and support over the duration of this study. The knowledge and motivation that Patrick provided was invaluable, and I could not have asked for a better supervisor for this research.

I thank the rest of my proposal committee, Prof. Deshen Moodley and Dr. Geoff Nitschke, for their help establishing the focus of the thesis. I also thank the members of the Zamani project, in particular Prof. Heinz Rüther, for showing me the importance of this research and allowing me to use their data.

To my friends in the Computer Science department that I have met while at the University of Cape Town, thank you for the conversation, advice and laughter that created an enjoyable learning environment.

I also thank my mother, sister and brother-in-law for their ongoing support that I could not have completed this study without. Lastly, I thank my late father, who taught me the value of education and to always persevere.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cultural heritage preservation involves protecting objects of cultural significance by maintaining their original condition for as long as possible [124]. In 1972, the United Nations Educational, Scientific and Cultural Organization (UNESECO) identified cultural heritage preservation as a necessary means of transmitting cultural heritage to future generations [122]. There are over 1070 UNESCO world heritage sites [6] and cultural heritage preservation is a large field with applications including the preservation of art, architecture and archaeology.

FIGURE 1.1: Photograph of Petra treasury (left), and 3D model generated from terrestrial laser scanning (right) [5].

Digital cultural heritage preservation uses various technologies to record and store objects. This way, a digital version of the object can be used for research, education or restoration if the original object degrades or is destroyed. For example, the University of Cape Town's Zamani project [5] have used terrestrial laser scanning (TLS) to record

over 50 cultural heritage sites in Africa and other continents. These recordings are stored as *point clouds* (large collections of 3D points) which are later used to generate high-resolution 3D models such as the Petra treasury shown in Figure 1.1.

Due to their size and varied contents, preparing point clouds for preservation requires a large amount of time. For example, to retain historical accuracy, Zamani use point cloud editing software to manually remove unwanted points (e.g. vegetation, wires, vehicles) from their scans. Figure 1.2 shows a scan before and after it is cleaned of vegetation points. This task is time-consuming as it may take 30 minutes or longer to clean a scan, and campaigns can contain dozens or hundreds of scans.

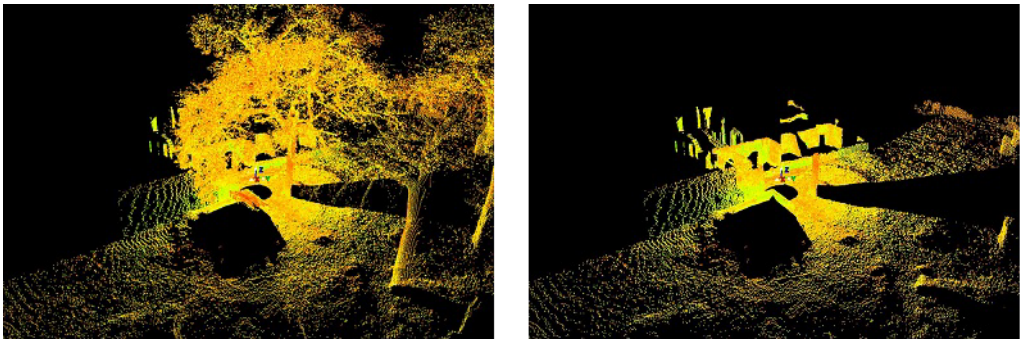


FIGURE 1.2: A raw terrestrial laser scan before (left) and after (right) the manual removal of vegetation points [107], with points coloured by signal intensity (green strong, red weak).

To address this problem, recent works based on machine learning have proposed semi-automatic methods for single scan cleaning [77] and incremental scan campaign cleaning [74], which reduce cleaning times to an extent. However, there is still a need for methods that accurately and automatically draw semantic information from an entire scan campaign's points in a single step, which could further accelerate tasks like scan cleaning.

This is only one example application. In addition to scan cleaning, machine learning methods that accurately categorise points in large and diverse terrestrial laser scans can help with tasks such as robotic navigation, urban planning and agricultural monitoring. The area of research that these machine learning methods belong to, and the focus of this thesis, is *point cloud classification*.

1.1 Point Cloud Classification

Point cloud classification is a machine learning approach to automatically predicting the *class* of points in point clouds. These predictions provide categorical information on points which can be used for further analysis or processing. For example, the classification of points in a street scan (Figure 1.3) can identify obstacles and be leveraged by self-driving cars for navigation. Or, as previously discussed, a point classifier can help scan cleaning operators identify and remove unwanted points faster by predicting points in a scan as *Keep* or *Discard*.

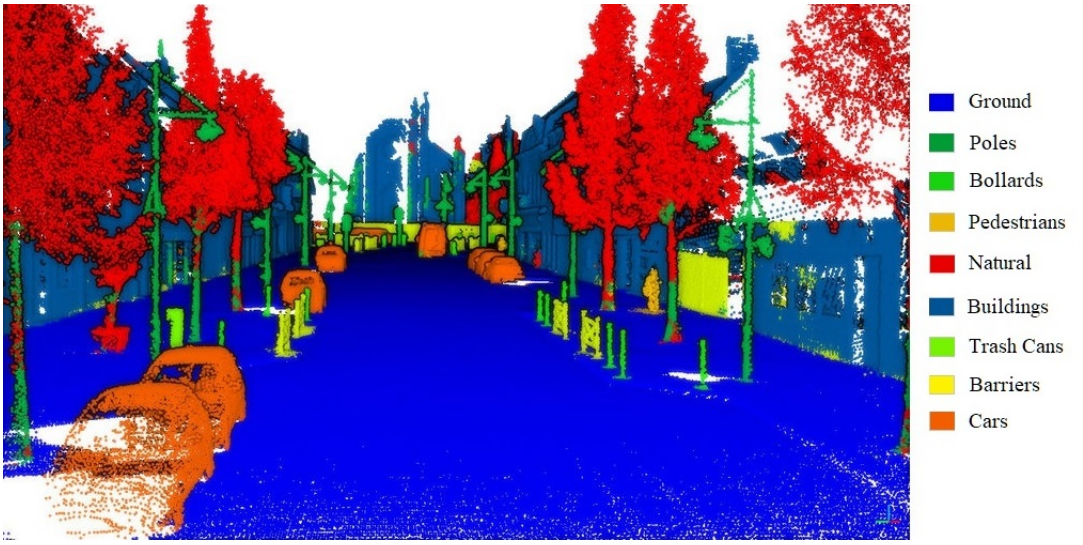


FIGURE 1.3: Example of classified point cloud, adopted from [104]. Points are coloured by their predicted class, as indicated by the legend.

A common approach to point cloud classification uses a *supervised* classifier, trained on *features* extracted from labelled points in a subset of campaign scans chosen for training. Once trained, the classifier can predict the class of points in the remaining, unseen scans of the campaign. This is a *campaign-based* approach, where two subsets of scans from the same site are used for training and prediction respectively. This approach is advantageous for terrestrial scan campaigns, which typically contain structures or objects unique to the site, as the classifier can learn from campaign-specific traits in the training set before classifying unseen scans. Another approach involves training general classifiers on external datasets. These classifiers, however, may not generalise well to diverse terrestrial laser scans containing structures the classifier has not seen before.

In either case, the speed and accuracy of a point cloud classification system greatly depend on which classifier and features are used, as well as the descriptiveness of the training data. When performing campaign-based classification, it is therefore important to choose a suitable classifier and feature set in addition to training scans that describe

unseen campaign scans well. Due to the importance and difficulty (discussed later) of these choices, this thesis investigates the use of various classifiers and feature sets when used for terrestrial scan campaign classification, and proposes a method that automatically selects representative training scans. More detail on these investigations and their motivation is provided below.

1.2 Proposed Solution and Motivation

To address the above challenge, a framework comprising various classifiers, feature sets and a new representative scan selection method was developed. Through the evaluation of this framework, this thesis investigates the performance of Random Forests (RFs), Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) and a variety of feature sets when classifying terrestrial laser scanning campaigns. The thesis also determines if the proposed automatic scan selection method can select a representative set of training scans that yield a higher classification accuracy than a randomly selected training set. These investigations are motivated by:

1. There are many different classifiers and point features available for point cloud classification, the use of which can have a large effect on predictive speed and accuracy. Choosing which of these to use is important, but identifying the best classifiers or features based on their success in the literature is difficult due to differing implementations, datasets and evaluation methods. The first aim of this thesis is therefore to to investigate the speed and accuracy of a selection of classifiers and point feature sets when used to predict diverse points from large terrestrial laser scanning campaigns.

2. Choosing which scans to label for training is also important, as the classifier's accuracy depends on the quality of the training set. However, manually determining the most suitable training scans is not straightforward, and inspecting scans is increasingly onerous for larger datasets. Randomly selecting scans is fast, but does not guarantee a representative training set. Furthermore, a preliminary survey of the literature did not reveal any existing methods that automatically select scans suitable for training. The second aim of the thesis is therefore to propose an approach to automatic, representative training scan selection and to determine if scans selected by the method yield a more accurate classifier than random selections when used for training.

This research could help users working with terrestrial laser scans, such as the Zamani project, by identifying methods that automatically provide accurate point information

and reduce the need for manual tasks like cultural heritage scan cleaning. If successful, the proposed scan selection method could reduce data preparation times and improve predictive accuracy by automating the task of training scan selection. Lastly, the direct comparison of existing classifiers and features can inform other research in the area, while the proposed scan selection method can form a basis for similar methods in the future.

1.3 Research Questions

Based on the proposed solution and its motivation, two research questions are asked to focus the research. These questions are:

1. *Which of a selection of classifiers (Random Forest, Support Vector Machine or Multi-Layer Perceptron), and a variety of point feature sets, achieves the fastest speed and highest accuracy when classifying points from large and diverse terrestrial laser scanning campaigns?*

   Different machine learning algorithms, such as Random Forests, Support Vector Machines and Multi-Layer Perceptrons, have been used to classify point clouds generated from various sources. These algorithms use sets of features extracted from points for both training and prediction. It is likely that some classifiers and feature sets perform better than others depending on the type of data, scene and recording equipment used. This research question asks which of these three classifiers and a variety of feature sets are the fastest and most accurate when classifying, specifically, diverse point clouds generated from large terrestrial laser scan campaigns.

2. *How do we automatically select a representative set of scans from a scanning campaign that, when used for training, yields a classifier with higher accuracy than a classifier trained on a random selection of scans?*

   The accuracy of any classifier is dependent on the quality of its training data. For scan campaign classification, it is therefore important that the scans chosen for labelling and training describe the other scans in the campaign well. However, identifying suitable scans is difficult and time-consuming, and randomly selecting scans (although fast) leaves the accuracy of the classifier to chance. This question asks how a method can automatically identify scans that, when trained on, yield a classifier that is more accurate than one trained on randomly selected scans.

1.4 Scope and Limitations

The focus of the thesis is on methods that contribute towards the training and classification of points in scans. Post-processing tasks that occur after classification, such as spatial regularisation and scan cleaning, are out of scope. However, the presented findings and framework are intended to improve the results of such tasks.

There are many classifiers and features that can be used for point classification. To keep testing feasible, only a selection of these, based on a review of the literature, are evaluated. For scan selection, the proposed automatic method is evaluated against random selection to provide a preliminary understanding of the method's performance. Comparison to manual selection is out of scope due to time constraints as it would involve lengthy user tests and teaching users how to identify representative scans. In addition, selections made by users may still be effectively random on larger datasets. However, comparison to manual selection could be explored in future work after the proposed method's performance is better understood and its design is refined.

Lastly, testing is limited to four datasets: two cultural heritage scan datasets with *Keep* or *Discard* labels, and two other terrestrial laser scanning datasets with multiclass labels. This allows the methods' performance and robustness to be sufficiently evaluated within a reasonable amount of time.

1.5 Overview of Thesis

The remainder of this thesis is separated into seven chapters. Chapter 2 introduces background theory on point clouds, segmentation and machine learning as these form the foundation of the framework developed to answer the research questions. Chapter 3 presents a high-level overview of the framework and a description of its three core components: (1) point features and feature selection, (2) point classification and (3) scan selection. The next three Chapters (4, 5 and 6) are dedicated to these components, each including a literature review and a detailed description of their component's implementation. Chapter 7 is split into two sections: experiments and results. The experiments section describes how the components were tested, while the results section analyses the results of the experiments in depth. Lastly, Chapter 8 concludes the thesis and discusses future extensions and applications.

Chapter 2

Background

The previous chapter introduced the problem of 3D scan campaign classification and identified the aims of the research. This chapter provides the background theory needed to understand the proposed solution. This includes theory on point clouds, segmentation and machine learning as these are core concepts behind the component's of the framework. Later chapters on point cloud classification, features and scan selection provide more detail on the concepts introduced below.

2.1 Point Clouds

A point cloud is a set of points defined by a coordinate system. A 3D point cloud contains points with three spatial coordinates (X, Y and Z), which describes their position in 3D space. Depending on how the point cloud is created, the points may be appended with colour or intensity information. An example of a raw point cloud with per point colour is shown in Figure 2.1. Raw point clouds can be processed to generate 3D meshes and models for further analysis or application.

When creating point clouds from real-world objects, a *LiDAR* (Light Detection And Ranging) scanner is used. This technology was first used by meteorologists in the 1960s to measure natural clouds [46]. LiDAR scanners are now used for both airborne laser scanning (ALS) and terrestrial laser scanning (TLS), can be either stationary or mobile, and have applications in multiple fields including geography, autonomous navigation, seismology and geomatics [23].

FIGURE 2.1: Example of a raw point cloud with colour values for each point, captured with a LiDAR scanner. Adopted from Autodesk tutorial [1].

When scanning large areas, LiDAR scanners based on the time-of-flight principle [52] are typically used as they can record points at long distances (up to 1 kilometer, depending on the model). Using this principle, the scanner probes the subject with a laser light and times the round-trip of each pulse of light. Given that the speed of light c is known, the distance to the surface the laser probed may be calculated as $c * t/2$ where t is the round-trip time of the pulse. Since the scanner can only detect points in one direction at a time, modern scanners use rotating mirrors to scan in different directions.

While time-of-flight LiDAR scanning is a popular method it has its strengths and weaknesses [15]. It can capture points over long distances, but its accuracy is lowered due to the difficulty of calculating the round-trip time precisely. In addition, when the laser hits the edge of an object the resulting point can be in the wrong place due to the averaging of two different locations of one pulse, leading to a scattered point effect. Furthermore, the density of the point cloud decreases as the distance to the scanned object increases. Lastly, when scanning at higher resolutions (requiring minutes) any movement or change in lighting can create distorted scans. High resolution scans can also detect dust and other particles in the air.

[1] Autodesk point cloud tutorial available at Autodesk website.

2.1.1 Point Cloud Features

A feature is a piece of information that describes an object or a part of an object. They can be extracted from and used to describe different types of data such as audio, sentences, images and point clouds. Descriptive features enable tasks such as segmentation and classification (discussed later in this chapter).

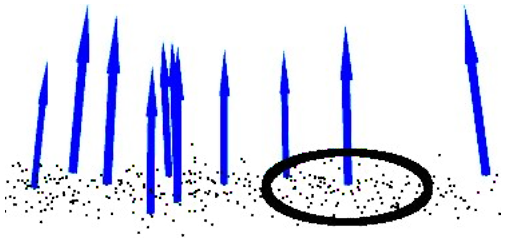


FIGURE 2.2: Example of a normal feature (blue arrow) describing a point's local neighbourhood (black circle), from [2].

Point cloud features describe either a point's local neighbourhood or the global properties of an entire point cloud. While global features are useful when describing isolated regions and segmented objects as in [105], local features are more practical for point clouds containing dispersed points that are not easily segmented, such as those generated by terrestrial laser scanning. A simple example is the normal feature illustrated in Figure 2.2 above, which describes the direction of the vector perpendicular to a point neighbourhood's surface. More features are discussed in detail in Chapter 4.

2.2 Segmentation

Segmentation is the process of partitioning data into multiple homogeneous groups that share some property. It is a general concept used across multiple fields such as economics, marketing and biology as well as for several computing tasks including computer vision, natural language processing and network segmentation. In computer vision, segmentation involves partitioning sets of image pixels into super-pixels or sets of of voxels (such as points in point clouds) into super-voxels.

The pixels or voxels in each region are similar with respect to some property or feature. For example, pixels in the same region could have similar colour as shown in Figure 2.3. After an image or point cloud is segmented, adjacent regions should be significantly dissimilar with respect to these properties [116]. Due to this simplified and meaningful representation, the segments can be more efficiently analysed [12] by tasks such as classification.

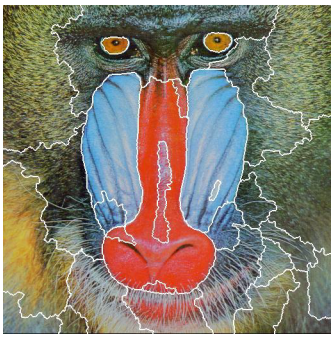


FIGURE 2.3: Image segmented into regions using pixel colour.

2.2.1 Point Cloud Segmentation

There are a variety of approaches to point cloud segmentation. The most commonly used methods can be grouped into four categories: model-fitting, region-growing, graph-based and feature-clustering. Similarly to image segmentation, the goal of point cloud segmentation is to segment the point cloud into groups of similar points, as shown in Figure 2.4. An explanation of these approaches and some usage examples is given next.

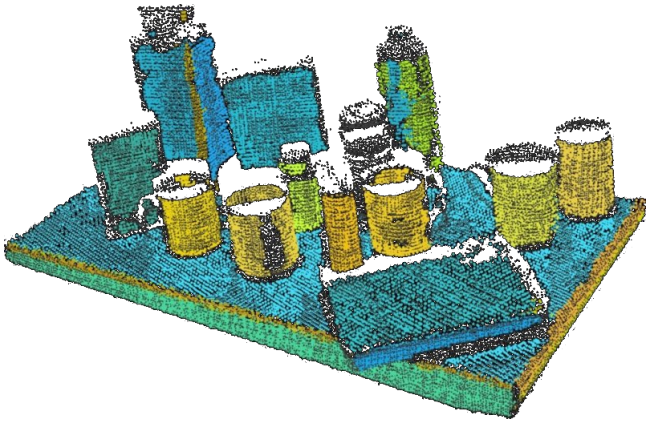


FIGURE 2.4: Example of segmented point cloud from [3], with colour denoting points in the same segment.

Model-fitting methods

Model-fitting methods work by attempting to group points into geometric shapes such as spheres, cylinders, planes and rectangles. If a set of points are found to share the same mathematical representation of a given shape, they form a segment.

The best-performing model-fitting method, RANdom SAmple Consensus (RANSAC) [37], has been used for tasks such as plane extraction [108] and polyhedral rooftop extraction [19]. While model-fitting methods can be fast and accurate for small datasets and simple shapes, they become slower and less accurate as the dataset and scene complexity grows [79]. They are also sensitive to point cloud noise and low density.

Region-growing methods

Region-growing methods involve combining neighbouring points with similar properties which consequently divides dissimilar regions into segments. There are two types of region-growing methods: seeded (bottom-up) and unseeded (top-down). Seeded methods start by forming regions around a selected number of seed points until a certain criterion is met. Unseeded methods do the opposite by starting with one large region containing all points and dividing it into smaller regions so long as a threshold is unsurpassed.

Region-growing methods have been used to segment surfaces of large buildings [123], recognise potential building features [94] and segment planar regions in sparse point clouds [20]. While they perform well on certain datasets, seeded methods are limited due to their sensitivity to seed point positioning and compatibility thresholds [79]. Unseeded methods are also limited by their dependency on prior knowledge (e.g. how many regions to split).

Feature-clustering methods

Feature-clustering methods form segments based on grouping extracted point features. After calculating feature vectors from point neighbourhoods, a cluster analysis algorithm such as k-means groups nearby points with similar feature descriptors.

Feature-clustering methods have been used to segment terrestrial laser scans [34] and cluster laser data surfaces [36]. One feature-clustering method, Voxel Cloud Connectivity Segmentation (VCCS) [85], uses geometric relationships between points in addition to colour and spatial features to *over-segment* point clouds as seen in Figure 2.5. An over-segmented point cloud is made up of regions where the points within a single region are similar but points in adjacent regions are not necessarily dissimilar.

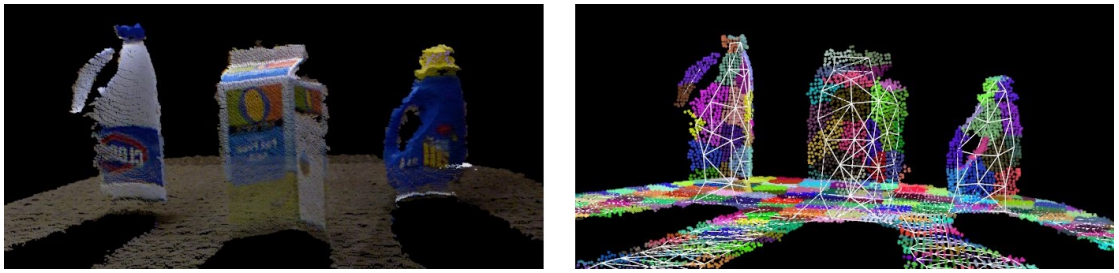


FIGURE 2.5: Original point cloud (left) and its over-segmentation (right) using the VCCS algorithm, from [1].

Feature-clustering methods are more robust than region-growing methods [123]. However, they are limited by their dependence on high-quality features, and are computationally expensive when calculating multi-dimensional features in large datasets.

Graph-based methods

Graph-based segmentation methods operate by treating the point cloud as a graph. In this graph, each vertex corresponds to a point in the cloud and the edges connect pairs of adjacent points.

Notable graph-based methods such as Markov Random Fields (MRF) [26] and Conditional Random Fields (CRF) [60] have been used to segment points in complex urban scenes [109] and segment points with different geometric surfaces [106] respectively. These methods can segment complex, noisy point clouds with uneven density. However, they are of higher complexity and are usually slower than previously discussed methods.

2.3 Machine Learning

Machine learning algorithms improve at a given task by adapting to data rather than following an explicit set of instructions. The core objective of a learning algorithm is to generalise from experience [14], meaning to improve at a task on new, unseen data after experiencing learning data.

Machine learning is most commonly used in areas where finding an optimal solution to a problem through explicit programming is not obvious or is infeasible [110]. This includes tasks such as speech recognition, spam filtering, fraud detection and several computer vision tasks including object recognition, robotic navigation and medical imaging diagnosis.

There are two main categories that machine learning algorithms typically fall into; supervised and unsupervised [75]. This distinction depends on whether the training data is labelled or not. With supervised learning, the algorithm has access to both the input data and their desired output labels. For example, in the case of image recognition the algorithm would learn from a set of images and a label for each such as “indoor” or “outdoor” before predicting the class of unseen images. This is an example of a binary classification since there are only two possible labels - this is different from multiclass classification where there are three or more possible labels. Unsupervised learning methods are not given output labels, they instead are used to recognise patterns in unlabelled data. An example of this is feature learning where an algorithm discovers features needed to describe samples, or cluster analysis where unlabelled samples are assigned to groups according to some similarity metric.

In recent years, *deep* learning methods have made major advances in solving difficult problems and have seen a significant increase in popularity. Deep-learning [65] methods learn a representation of training data through multiple levels of transformation. Each

level transforms the data into a higher and slightly more abstract level, obviating the need for feature engineering as features are learned from training data directly. While deep-learning methods have achieved state of the art results in many benchmarks, most implementations require very large amounts of training data and computation time.

2.3.1 Point Cloud Classification

Point cloud classification typically uses supervised learning to predict the class of (classify) points in the cloud. Due to the increased availability of consumer depth cameras like the Microsoft Kinect and advances in professional LiDAR equipment, point clouds are a popular and effective way of recording 3D representations of real world scenes. Following this, point cloud classification has become an active area of research and an important task in computer vision. Various approaches with different machine learning methods at their core have been proposed. While the performance of these methods greatly depends on the problem's context and the quality of the training data, certain methods have been shown to perform better than others in most conditions leading to a rise in their popularity.

The next three sections discuss three successful and popular machine learning models used for point cloud classification: Support Vector Machines, Random Forests and Artificial Neural Networks. This provides the background theory necessary for the point classification chapter.

2.3.2 Support Vector Machines

Support Vector Machines (SVMs) were originally limited to non-probabilistic binary classification [22], and could therefore only distinguish between two classes. Several extensions have been made to SVMs to enable multiclass classification.

SVMs treat training samples feature vectors as points in space that are mapped such that samples from different classes are as far away from each other as possible. Specifically, they use a kernel function to map each training sample's feature vector into a higher-dimensional space, and find hyper-planes that fit between the training samples, as shown in Figure 2.6. The goal is to find the hyper-plane that maximises the margin between the nearest feature vectors from each class (called support vectors). The support vectors are the only vectors that affect the position of the hyper-plane, and therefore define the decision function.

When the SVM receives a test sample, it maps it to the hyper-space and classifies it based on which side of the max-margin hyper-plane it falls into.

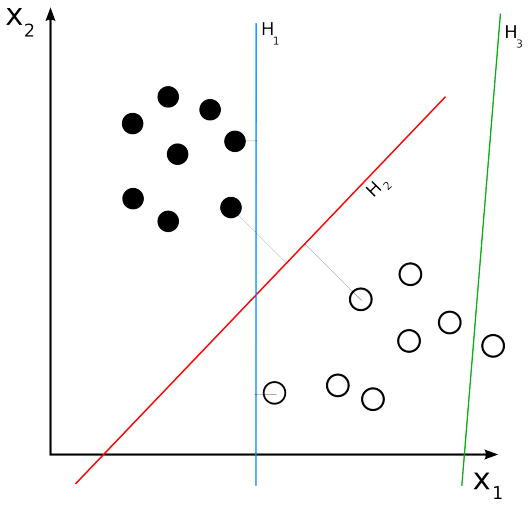


FIGURE 2.6: SVM hyper-planes H_1 - H_3 (blue, red and green lines) separating training samples (circles). Hyper-plane H_2 maximises the margin between the nearest samples from each class.

Multiclass SVM classification approaches can be categorised as indirect or direct. The indirect approach involves training multiple SVMs, either in a one-against-rest or one-against-one manner [125]. When using a one-against-rest k-class classifier, k SVMs are trained to separate each class from the all other classes and test sample prediction is determined by which SVM gives the maximum output value. When using a one-against-one k-class classifier, all possible $k(k-1)/2$ pairwise SVMs are trained and prediction is determined by which class receives the most votes. Direct or *all-together* methods consider all classes at once, however their practical use is limited due to their high computational complexity [125].

2.3.3 Random Forests

Before discussing random forests (RFs), it is necessary to describe their basis - decision trees. A decision tree is a data structure consisting of leaves and branches where leaves represent class labels, and branches represent the splitting of samples (based on considered features) that lead to a given class label [69]. Decision trees are trained recursively starting at the top of the tree, called the root. The root node splits features into two child nodes based on a metric such as the Gini index [45]. This is repeated until a stopping condition such as the maximum depth of the tree is met, or if splitting further would have no noticeable improvement.

To predict the class of an input feature vector, the vector is given to the root node. For each non-leaf node, the procedure moves to the left or right child depending on the value of the features whose indices are stored at the current node. This continues until a leaf

node is reached and the input vector is classified as the class assigned to the leaf node during training.

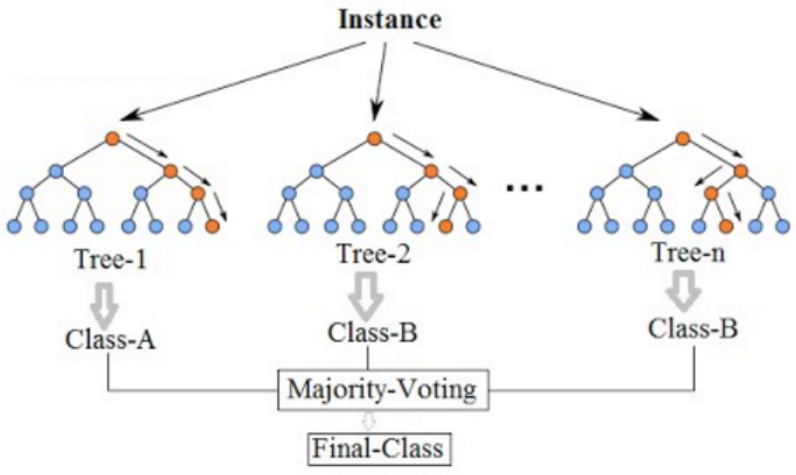


FIGURE 2.7: Illustration of RF prediction from [4], where an instance is classified by the majority vote of an ensemble of n decision trees.

Random forests [17] are an ensemble method that combines multiple decision trees into a more powerful predictor. They classify input vectors by giving each tree in the forest the vector for classification and then bagging the resulting outputs to determine the majority vote and the final output as shown in Figure 2.7.

When trees in the forest are trained, they are given the same parameters but use different training sets. These training sets are randomly selected (with replacement) from the original dataset and are the same size as the original dataset. The resulting datasets can therefore contain some vectors more than once while other vectors may be absent. To further randomise, when a node is split it only considers a random subset of the training features. A different subset is used for all nodes, but they have the same size which is typically set to the root of the number of variables. Due to this randomisation and averaging of loosely correlated decision trees, RFs have lower variance than single decision trees [41].

2.3.4 Artificial Neural Networks

Artificial neural networks (ANN), loosely inspired by the biological neural networks of brains [54], are machine learning models formed by collections of connected nodes or *neurons*. These artificial neurons can receive signals, process them and then generate an output signal for connected neurons to use as input.

A common ANN is the Multi-Layer Perceptron (MLP). An MLP consists of one input layer, one output layer and one or more hidden layers. These layers contain one or more neurons that are connected with the neurons in the previous and next layer. Figure 2.8 below illustrates a three layer MLP with three inputs, two outputs and a hidden layer containing three neurons. In recent years, *deep* neural networks with many hidden layers have been used to learn representations from training data, obviating traditional feature extraction, and classify point clouds [96][63]. More information on these networks is given in Section 5.1 of the classification chapter.

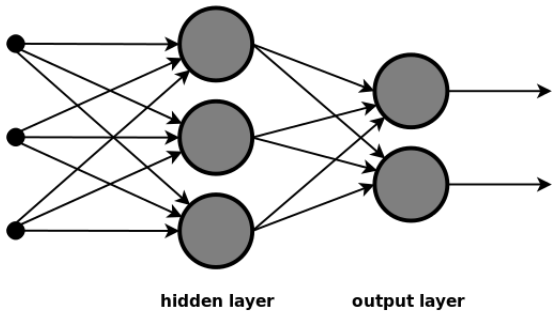


FIGURE 2.8: An MLP with three input neurons, two output neurons and a hidden layer with three neurons.

Each neuron in the MLP has input and output links. The input links pass the neuron the values from neurons in the previous layer to process and the output links pass the neurons response to the neurons in the next layer. When a neuron receives a value from the previous layer, they are weighted and summed up along with a bias term. This sum is then transformed using an activation function f such as symmetrical sigmoid.

Given the outputs x_j of the layer n, the outputs y_i of the layer $n+1$ are computed as:

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1} \tag{2.1}$$

$$y_i = f(u_i) \tag{2.2}$$

To train the network, the input layer is first set to the same size as the number of features in the training feature vector. For each training vector, the values of the vector are taken as input and passed to the first hidden layer. Each neuron in the hidden layer then computes their outputs using their stored weights and activation functions. The outputs are passed to the next layer until the output layer is computed. This is done for every training sample via backpropagation, where weights are iteratively adjusted with every pass. Once completed, the network can predict the response of test samples by the output given when passing the sample's feature vector through the network.

2.3.5 Cluster Analysis

Clustering is an machine learning approach where samples are assigned to groups (called *clusters*) such that samples in the same group are more similar to each other than samples in other clusters. Clustering is typically unsupervised i.e. does not require the samples to be labelled. The metric used to define similar samples, and therefore the way that clusters are formed, depends on the clustering algorithm. The most prominent clustering methods are discussed below.

Centroid-based algorithms such as k-means [70] represent clusters as their most central (mean) feature vector. They aim to find k clusters such that the squared distances of the cluster members to their centroid is minimised. Figure 2.9 shows the clustering results of a simple 2D dataset after applying k-means with k=3. There are variants of the k-means algorithm including k-medoids [57] where centroids must belong to the dataset, and fuzzy c-means [29] where points can belong to multiple clusters.

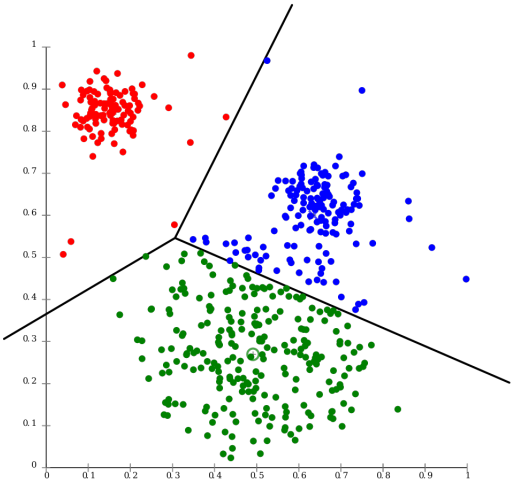


FIGURE 2.9: k-means clustering of 2D points into k=3 clusters denoted by red, green and blue.

Other approaches include density-based, connectivity-based and distribution-based clustering. Density-based methods, such as DBScan [32] and OPTICS [10], define clusters as regions with a high density of points while points in low density areas (deemed noise) are not assigned to a cluster. Connectivity-based methods form clusters of points that are near each other and assumed to be similar. This is done repeatedly with varying distance thresholds, resulting in a visualised hierarchy of cluster assignments for the user to select from. Lastly, distribution-based clustering methods use statistical distributions to cluster points. One example is Gaussian mixture models [100] where several random

Gaussian distributions are initialised and samples are clustered by which distribution they most likely belong to.

This chapter has introduced the theory needed to understand the design and implementation of the methods that form the scan campaign classification framework. The next chapter provides a high-level description of the framework's architecture and the role of each component.

Chapter 3

Framework Design

This chapter provides a high-level overview of the scan campaign classification framework's design. The framework is composed of three core components: (**1**) Point Features and Feature Selection, (**2**) Point Classification and (**3**) Representative Scan Selection.

First, the framework's architecture is presented in Section 3.1 to illustrate the connection of these components. Then, the role and elements of each component are described in Sections 3.2 - 3.4. This overview provides context for later chapters where method choice and implementation details are explained.

3.1 Architecture

The framework is designed to enable experiments that answer the research questions on features, classifiers and scan selection. As discussed in the introductory chapter, these questions are:

1. *Which of a selection of classifiers (Random Forest, Support Vector Machine or Multi-Layer Perceptron), and a variety of point feature sets, achieves the fastest speed and highest accuracy when classifying points from large and diverse terrestrial laser scanning campaigns?*

2. *How do we automatically select a representative set of scans from a scanning campaign that, when used for training, yields a classifier with higher accuracy than a classifier trained on a random selection of scans?*

The framework is split into three core components containing the methods needed to address these questions. Together, the components form a complete point cloud classification system that reads labelled point clouds and outputs the predicted classes of

points in unseen test clouds. Through development and configuration of the framework, different methods can be tested and evaluated to answer the research questions. The diagrams in Figure 3.1 (Training Phase) and Figure 3.2 (Prediction Phase) illustrate the component pipeline during training and prediction.

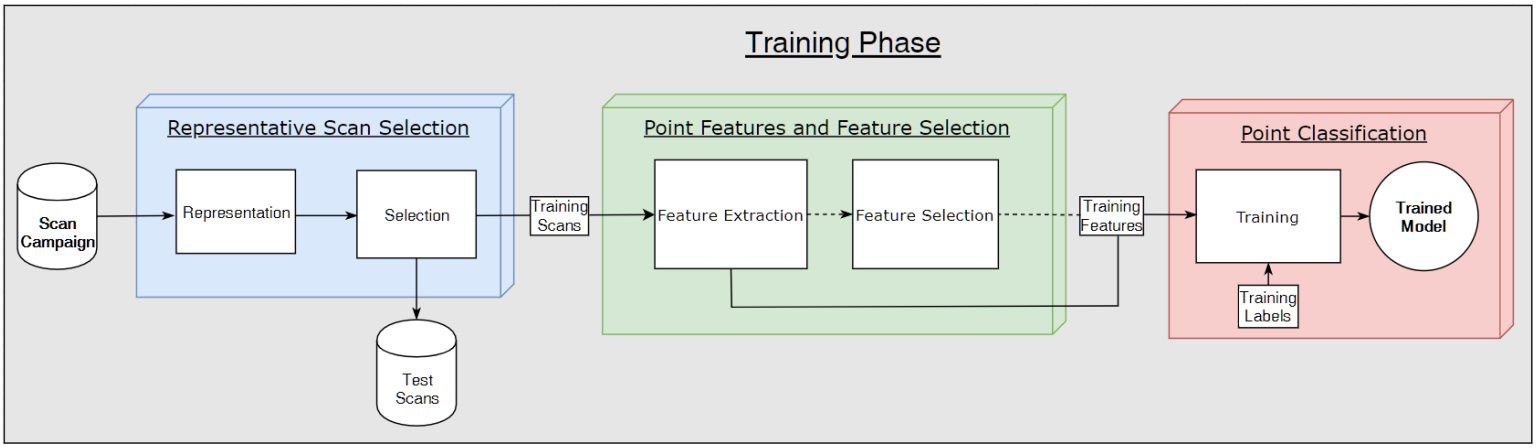


FIGURE 3.1: Component pipeline during training. Training starts with the *Scan Campaign* and ends with a *Trained Model*. *Test Scans* are set aside for the prediction phase that follows. All three components are used for training.

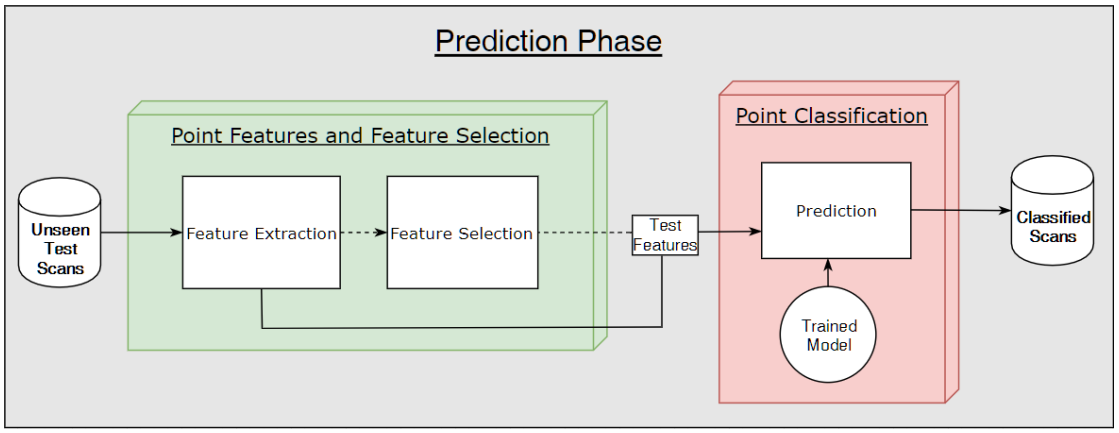


FIGURE 3.2: Component pipeline during prediction. Prediction starts with the *Unseen Test Scans* and ends with a set of *Classified Scans*. Only two components are used for prediction as scans are selected during the training phase.

The sections that follow explain the role of each component in the framework and give a brief description of their methods. More detail on these methods is provided following a review of the literature in each component's chapter.

3.2 Point Features and Feature Selection Component

The purpose of this component is to extract features from points in scans. These features are later passed to the classification component where they are used for training or prediction. The component also contains feature selection methods, which identify a subset of the most useful (descriptive) features.

Feature Extraction
Point features describe properties of the points in a scan. The component extracts multiple features in order to capture enough information for a classifier to learn from. These features describe the shape, size and other properties of a point's local neighbourhood(s). Features from point clouds re-sampled to different resolutions are also extracted.

Feature Selection
Two feature selection methods form this part of the component. They represent two approaches to feature selection: classifier-dependent and classifier-independent. The former utilises a classifier to determine feature importance, while the latter uses statistics to rank features. A subset of the highest importance or top-ranking features form a reduced feature set.

3.3 Point Classification Component

The classification component uses previously extracted point features paired with class labels to train one of three classifiers. The classifier then predicts the class of unseen scan points, and generates classified point clouds as well as speed and accuracy statistics.

Training
The component includes three classifiers for training. These are a random forest (RF), Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP). Although internally different, the classifiers accept the same training data (feature vectors and class labels). From this data, the classifier learns a model which is stored for the prediction phase.

Prediction
To predict the class of points in unseen scans, the points' feature vectors are passed to the previously trained classifier. After processing these vectors, the classifier's predictions are used to generate a classified point cloud with class colours. If ground truth labels exist, a number of statistical measurements are also made.

3.4 Representative Scan Selection Component

This component attempts to automatically select the most descriptive scans in a scanning campaign for training, such that the accuracy of the classifier is maximised when predicting other scans in the campaign. To do this, the component generates rough representations of each scan which are used to score and select scans according to specific criteria.

Representation

The component uses a series of steps to generate a representation of each scan in the campaign. These include segmentation, feature extraction and clustering. In essence, the points in each scan are segmented, and all segments from all scans are clustered. A scan's representation is then derived from its segments' cluster assignments.

Selection

The component includes three scan selection *schemes*, each designed to meet different criteria. Using the derived representations to score and rank scans, they identify the most *balanced*, *similar* or *distinct* scans in the dataset. As with other methods discussed above, more detail on the schemes is provided in the component's relevant chapter. A subset of the highest ranking scans then form the training set for the remainder of the pipeline. Consequently, scans not selected for training form the set of test scans used during the prediction phase.

This short chapter has provided a high-level overview of the scan campaign classification framework's design and a description of each component's role. The next three chapters are dedicated to the three components of the framework, and include a literature review of related works as well as the design and implementation details of each component.

Chapter 4

Point Features and Feature Selection

Point features are foundational to the scan campaign classification framework. It is therefore important to select features that are relevant to both the given dataset and classification task. This chapter introduces the first of the framework's core components: Point Features and Feature Selection. First, existing features and selection methods in the literature are reviewed in Section 4.1 to identify suitable techniques. Section 4.2 then details the implementation of the techniques chosen for the component, and methods needed for integration with the framework.

4.1 Related Work

Before implementing the features component, it is necessary to survey existing methods. This review is split into two sections: point features and feature selection. The point features section discusses existing features, while feature selection looks at methods for selecting descriptive features. The differences between the feature selection methods, as well as their advantages and disadvantages, are then discussed.

4.1.1 Point Features

There are many point features in the literature, the use of which depends on the scene and scanning method. For example, synthetic computer-aided design (CAD) point clouds, an indoor room recorded with a Kinect, an urban street recorded with a mobile laser scanner and a rural area surveyed via aerial scanning typically require different feature

sets for accurate classification. These sets depend on the quality of the training data, how similar the classes to be classified are, as well as the differing densities, shapes, lighting, colour uniformity and noise of points in the scenes. This review is therefore limited to features that have been used to classify points in terrestrial laser scans as they are the focus of this research.

While some features use only a single point's information, most features are based the point and other nearby points i.e. the point neighbourhood. A common way to find neighbourhoods is to search for the k closest points using the k-nearest neighbour (k-NN) algorithm [39]. There are also spherical [66] and cylindrical [35] neighbourhoods, formed by all points found within a sphere or cylinder of a fixed radius r. Depending on the density of the point cloud and the k or r parameter, the speed of neighbourhood extraction varies greatly.

Colour and intensity features are sometimes available depending on the scanner. Along with 3D co-ordinates, each point in the cloud can have red, green, blue and intensity (RGBI) values. Intensity measures the optical strength of the laser signal returned while scanning. While colour features can be useful in ideal conditions e.g. synthetic CAD-based objects, they are often unreliable in real-world scenes due to changes in lighting caused by shadows, weather or time of day. Intensity is also unreliable due to the effect that scanning distances and angles have on the reflectance of objects [92].

Geometric features are computed from points' 3D co-ordinates. For example, a point's Z-value describes its height if the scanner is perpendicular to the XY plane while recording. The difference Δ between the smallest and largest Z, as well as the standard deviation σ of Z, in the point's local neighbourhood can help distinguish points belonging to short and tall objects. Other height-based features like vertical range $Z_{max} - Z_{min}$, height below $Z - Z_{min}$ and height above $Z_{max} - Z$ [50] are extracted from cylindrical neighbourhoods to describe thin vertical objects well. Raw X and Y co-ordinates are not suitable features for classification as they are relative to the scanner's position, and can therefore vary greatly despite belonging to points of the same class.

Features such as the radius of the sphere encapsulating a point's k-nearest neighbours, as well as the density of the neighbourhood, can also be extracted [128]. These are useful for differentiating, for example, scanner noise (large radius, low density) from valid points (small radius, high density).

Covariance features [133] describe the shape of the point neighbourhood. They are estimated after computing the eigenvectors and eigenvalues ($\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$) of the point neighbourhood's covariance tensor [13]. Many features can be derived from eigenvectors and eigenvalues. For example, the eigenvector V_3 with the smallest eigenvalue λ_3

is used to estimate the point's surface normal, while the neighbourhood's change of curvature is computed as $\lambda_3/(\lambda_1+\lambda_2+\lambda_3)$. Other shape features derived from these eigenvalues include planarity $(\lambda_2-\lambda_3)/\lambda_1$ and linearity $(\lambda_1-\lambda_2)/\lambda_1$. These are useful when comparing e.g. points from a pole to a wall; while points on a pole have high linearity and low planarity, points on a wall have low linearity and high planarity. Other features derived from eigenvalues include sphericity λ_3/λ_1, omnivariance $\sqrt[3]{\lambda_1\lambda_2\lambda_3}$, anisotropy $(\lambda_1-\lambda_3)/\lambda_1$, eigenentropy $-\sum_{i=1}^{3}\lambda_i ln(\lambda_i)$ and the sum of eigenvalues $\lambda_1+\lambda_2+\lambda_3$. First and second order moments of the neighbourhood around eigenvectors V_1 and V_2 have also been used to help identify boundaries between occluding objects [50].

2D features are extracted from the projection of the point cloud onto the XY plane. These features are useful in scenes with symmetrical or orthogonal objects like buildings. When projected onto a plane, a building wall becomes a line which is easily described by shape features. Previously described 3D geometric features such as radius and local point density [64] can be adapted into features extracted from circular 2D neighbourhoods [127]. Similarly, covariance features can also be adapted to use the eigenvalues derived from the circular neighbourhood's covariance tensor.

Neighbourhood Context

A limitation of local features is their lack of contextual information. As they are local, they do not describe the points outside of their neighbourhood. If included in the neighbourhood, some of these points could increase the descriptiveness of the features.

One solution is to increase the size of the neighbourhood by increasing k for k-nearest neighbourhood search or increasing r for spherical or cylindrical neighbourhoods. This can be done manually or automatically as in [129] where a single neighbourhood of optimal size k is found. While increasing the size of the neighbourhood broadens its context, it also increases neighbourhood and feature extraction times.

A different method of increasing context re-samples the neighbourhood's search space [87] in order to describe the local 3D structure across varying scales. In doing this, k remains fixed but the neighbourhoods capture a larger context as the search space becomes sparser.

Discussion

There are clearly many features to choose from when designing a scan campaign classification framework. Despite limiting the review to point features typically used with terrestrial scans, it is challenging to gauge their suitability to a given dataset. Some promising features and their strengths have been highlighted, but their performance may vary between datasets. The next section therefore looks at feature selection methods that could help identify a descriptive subset of scan campaign-specific features.

4.1.2 Feature Selection

Choosing the best set of features for a given dataset is not straightforward; a feature that performs well on one dataset or at one classification task may not necessarily perform well on others. Feature selection methods attempt to solve this by finding compact yet descriptive feature sets that improve computational efficiency without decreasing — and sometimes increasing — predictive accuracy. Notable feature selection methods, grouped by their classifier dependence, are described below.

Classifier-independent feature selection
Classifier-independent methods look directly at training vectors as well as their class labels to select features without the need of a classifier. They calculate a score for each feature which is used to rank and select the best features. They can be further divided into univariate and multivariate methods as in [127].

Univariate methods evaluate the relationship between features and classes to differentiate between relevant and irrelevant features. A score function calculates the relation between a vector containing all the observed values of a specific feature and its respective class label vector. A popular univariate score function is the Pearson correlation coefficient [88], which measures the strength of the linear correlation between a feature and a class between -1 and +1, where a strong negative (-1) or positive (+1) correlation indicates a relevant feature, and no correlation (0) indicates an irrelevant feature. Other notable score functions are Fisher score [38], Gini index [45], Chi-squared test [89] and information gain [97]. These measures and a brief description of what they indicate are summarised below:

- **Pearson correlation**: Indicates to which degree a feature is correlated with a class label.
- **Fisher score**: Calculates ratio between interclass and intraclass variance. A high ratio indicates a discriminative feature.
- **Gini index**: Measures dispersion or inequality which indicates a features ability to distinguish between classes.
- **Chi-squared test**: Assesses whether a class label is independent of a particular feature.
- **Information gain**: Reveals the dependence between a feature and a class label.

These methods can be combined by aggregating rankings from different score functions into one final ranking [127]. However, although univariate methods are fast to compute, they only consider the feature-class relationship and can still select redundant features.

Multivariate methods differentiate between relevant and irrelevant features like univariate methods do, but they also evaluate feature-feature relationships to identify redundant features. Unlike univariate methods, they consider multiple feature vectors simultaneously instead of one at a time. Multivariate score functions include ReliefF [59], Correlation-based feature selection methods [51][139] and Markov blanket-based methods [42][91]. As multivariate methods filter out redundant features, some of them have been shown to select features that out-perform the original feature set [127].

Classifier-dependent feature selection

Classifier-dependent methods interact with a supervised classifier to identify the best features. These features are optimised for the respective classifier, meaning they may not generalise to other classifiers as well as filter-based selections. However, if the same classifier is used for selection and prediction it can be more accurate than with features selected independently. Classifier-dependent selection is separated into wrapper and embedded methods.

Wrapper methods treat feature selection as a search task by building and comparing different combinations of features to each other. Feature sets are scored based on the accuracy they achieve when predicting a validation set.

One way to search for the best feature set is to find the most predictive feature and progressively add features that improve accuracy e.g. Sequential Forward Selection [134]. Alternatively, Backward Sequential Elimination starts with the entire feature set and successively deletes the worst performing features. Other search methods like best-first search [138] and stochastic random hill-climbing [118] have also been used. Although wrapper methods find features optimised for a specific dataset and classifier, the repetition of training and prediction steps makes them much slower than other methods.

Embedded methods determine the set of features that contribute the most to a classifier's accuracy as it is being trained. For example, the random forest algorithm [17] internally determines a "feature importance" measure for each feature based on its impact on the model's accuracy. Similar built-in mechanisms exist in the single decision tree algorithms ID3 [97] and C4.5 [98]. There are also embedded regularisation methods that minimise the size of feature coefficients and eliminate features with coefficients that are too close to zero [73].

Like wrapper methods, embedded methods interact with a classifier and can provide optimised selections that outperform features selected by classifier-independent methods. However, since embedded methods make their selection *during* training they are much faster than iterative and computationally expensive wrapper methods.

Discussion

A number of approaches to feature selection have been described. It is clear that they have their own advantages and disadvantages that should be considered before applying them.

Classifier-independent methods (univariate and multivariate filter methods) are easy to implement and perform faster than classifier-dependent methods since they do not require classifier training. Their resulting feature selections should also generalise well to different classifiers since they consider feature vectors alone and are not influenced by a specific classifier. Due to their speed, multiple filter methods can be combined to generate an aggregated ranking of features.

Classifier-dependent methods (wrapper-based and embedded), while not as general as independent methods, identify feature sets specifically chosen for their classifier. Although they are computationally expensive (especially wrapper-based methods) and slower than independent methods, their tailored feature selections should achieve higher accuracy when used to train their respective classifier.

The choice of feature selection method therefore depends on the requirements of the system. If speed and the ability to generalise are important, then classifier-independent methods are suitable. If slower speeds are acceptable and high accuracy with a specific classifier is required, then classifier-dependent methods are a better choice.

4.2 Implementation

This section describes the implementation of the features and feature selection component. The choice of features and methods used for the component is motivated by their applicability to terrestrial laser scans shown in the literature. The details of the component's implementation are given below. As a reference, a link to the repository of the framework's source code, including the features component, is provided in the appendix.

4.2.1 Point Features

Pre-processing

Before point features are extracted, the point cloud is downsampled to a density of one point per 2.5cm^3 using Point Cloud Library's octree class. Specifically, an octree is built from the original point cloud with a resolution of 2.5cm and a new point cloud is created from the XYZ averages of the points in each octree leaf. While this reduces the point cloud's level of detail, it mitigates the issue of non-uniform point density that occurs when scanning large areas. This improves the consistency of extracted features as it limits how much the features of points from the same class can vary due to density alone. In addition, downsampling the original point cloud results in much faster feature extraction.

Neighbourhood extraction

All of implemented point features are extracted from the point's local k-nearest and cylindrical neighbourhoods at three density scales. As discussed in the review of features, colour and intensity values are not used due to their unreliability, and the multi-scale neighbourhood approach is used to provide broader context.

The component uses PCL's `nearestKSearch` method to find a point's k=10 nearest neighbours. Rather than increasing k (and therefore search times) to improve the context of derived features, k remains constant and an approach based on multi-scale neighbourhoods [87] is employed. Three density levels, starting with an initial downsampling of 2.5cm, are used to extract point features. This is doubled twice to 5cm and 10cm, resulting in three different sets of k=10 nearest neighbours for each point.

A cylindrical neighbourhood search was also implemented to enable the extraction of additional height features. The implementation uses PCL's radius search on a k-d tree built from a projection of the 2.5cm density point cloud onto the x-y plane. The projection is fast (simply setting each point's Z to 0), but the radius search is increasingly

slow for larger radii due to the planar search space's high density. The cylindrical search is therefore limited to a radius of 5cm for all three density levels.

Feature extraction

A collection of 24 features are extracted at three scales (2.5cm, 5cm and 10cm), yielding a total of 72 features per point. The feature set contains 3D covariance, geometric and height features, as well as 2D variants of particular 3D features, chosen for their applicability to terrestrial laser scans and their usage in previous works [50][127]. The Eigen C++ library [47] computes the point's k-nearest neighbourhood eigenvectors (e_1, e_2, e_3) and corresponding eigenvalues ($\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$) needed for the covariance features. The geometric and height features are computed directly from k-nearest and cylindrical neighbourhoods. The majority of the covariance features were originally implemented in Octave based on the Matlab feature extraction code accompanying [127], but were later converted to C++ for improved performance and integration. All of the 24 implemented features' names, symbols and definitions are provided in Table 4.1.

In order to extract features at multiple scales, three k-d trees are generated during the original point cloud's three-stage downsampling. The point's 24 features are extracted in parallel from neighbourhoods found with the k-d trees (as well as the cylindrical neighbourhoods), and are combined to form the final 72 element feature vector.

4.2.2 Feature Selection

Two feature selection schemes were implemented: one classifier-independent and one classifier-dependent. The classifier-independent scheme combines univariate and multivariate filter methods while the classifier-dependent scheme leverages the feature importance values computed by an RF classifier. The schemes were chosen as they represent the two main approaches to feature selection found in the literature. Their implementation details are given below.

Classifier-independent selection

The classifier-independent scheme uses univariate and multivariate filter methods available in the scikit-learn (Python) library [90]. A total of six filter methods from the library are used to separately score and rank features before computing a combined ranking. Table 4.2 below gives the type of each filter selection method and a summarises what they measure.

The feature selection method loads vectors containing all 72 of the extracted training set features from a stored comma-separated values (`.csv`) file. Although this increases read

Name	Symbol	Definition
Covariance/shape features		
Verticality	V	$1 - \lvert\langle[001], \mathrm{e}_3\rangle\rvert$
Linearity	L_λ	$(\lambda_1 - \lambda_2)/\lambda_1$
Planarity	P_λ	$(\lambda_2 - \lambda_3)/\lambda_1$
Curvature	C_λ	$\lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
Sphericity	S_λ	λ_3/λ_1
Omnivariance	O_λ	$\sqrt[3]{\lambda_1 \cdot \lambda_2 \cdot \lambda_3}$
Anisotropy	A_λ	$(\lambda_1 - \lambda_3)/\lambda_1$
Eigenentropy	E_λ	$-\sum_{i=1}^{3} \lambda_i \cdot \ln(\lambda_i)$
1st Order Moment 1	M_1	$\sum_{i \in \mathrm{P}} \langle \mathrm{P}_i - \mathrm{p}, e_1 \rangle$
1st Order Moment 2	M_2	$\sum_{i \in \mathrm{P}} \langle \mathrm{P}_i - \mathrm{p}, e_2 \rangle$
2nd Order Moment 1	M_3	$\sum_{i \in \mathrm{P}} \langle \mathrm{P}_i - \mathrm{p}, e_1 \rangle^2$
2nd Order Moment 2	M_4	$\sum_{i \in \mathrm{P}} \langle \mathrm{P}_i - \mathrm{p}, e_2 \rangle^2$
Sum of EVs	$\Sigma_{\lambda 3\mathrm{D}}$	$\lambda_1 + \lambda_2 + \lambda_3$
Sum of EVs (2D)	$\Sigma_{\lambda 2\mathrm{D}}$	$\lambda_{2\mathrm{D}_1} + \lambda_{2\mathrm{D}_2}$
Ratio of EVs (2D)	$\mathrm{R}_{\lambda 2\mathrm{D}}$	$\lambda_{2\mathrm{D}_2}/\lambda_{2\mathrm{D}_1}$
Geometric features		
Radius	r_{3D}	$\mathrm{dist}(\mathrm{p}, \mathrm{P}_k)$
Density	D_{3D}	$(k+1)/(\frac{4}{3}\pi {r_{3D}}^3)$
Radius (2D)	r_{2D}	$\mathrm{dist}(\mathrm{p}_{2\mathrm{D}}, \mathrm{P}_{2\mathrm{D}k})$
Density (2D)	$\mathrm{D}_{2\mathrm{D}}$	$(k+1)/(\pi {r_{2D}}^2)$
Height features		
Height difference	$\Delta\mathrm{H}$	$z_{\max} - z_{\min}$
Height std. deviation	$\sigma\mathrm{H}$	$\sqrt{\sum_{i=1}^{k} (z_i - \overline{z})^2}/(k-1)$
Vertical range (cylinder)	$\mathrm{H}_{\mathrm{range}}$	$z_{\max} - z_{\min}$
Height above (cylinder)	$\mathrm{H}_{\mathrm{above}}$	$z_{\max} - z$
Height below (cylinder)	$\mathrm{H}_{\mathrm{below}}$	$z - z_{\min}$

TABLE 4.1: Extracted point feature names, symbols and definitions. $\boldsymbol{\lambda_i}$ and $\boldsymbol{e_i}$ are the ith eigenvalue and eigenvector derived from a covariance tensor of the points $\mathbf{P}$ in a point $\mathbf{p}$'s k-nearest neighbourhood. Features are extracted from 3D space unless stated 2D.

times, it is more straightforward than embedding Python filter methods in the C++ framework and allows other methods to use the same features at a later stage.

Each method generates a score for each feature. For most methods, a higher score denotes a better feature with the exception of the Gini coefficient where a lower coefficient is better, and the Pearson correlation where the score falls between -1 (strong negative linear correlation) and +1 (strong positive linear correlation). To account for this, Gini coefficient scores are ranked in reverse order and Pearson correlation scores are replaced with their absolute values.

After all of the scores are calculated they are ranked to form 6 separate rankings. The rankings (unique values between 1 and 72) are combined to form a single final ranking for each feature. The best 25 (lowest combined ranking) features are identified and their

Method	Type	Measures
Gini index	Univariate	Inequality or dispersion between features.
ANOVA F-value	Univariate	Amount of variance explained by a feature.
Mutual information	Univariate	Dependence between a feature and a class label.
Pearson correlation	Univariate	Correlation between a feature and a class label.
Fisher score	Univariate	Ratio between inter-class and intra-class variance.
Relief-F	Multivariate	Conditional dependencies between features.

TABLE 4.2: Implemented filter selection methods, their type and measurement.

indices are used to form a 72 element bit array, which is later used to reduce feature vectors before training or prediction. Although only 25 (approximately a third of all features) are chosen, such a reduction may still yield an accuracy close to the full feature set, and should enable comparison of different features and feature selection methods.

Classifier-dependent selection

The classifier-dependent scheme uses the Ranger [136] RF library. An RF is trained on all of the extracted features in the training set using the same settings as the RF in the classifier component (detailed later in Section 5.2) while additionally calculating feature *importances*.

The Ranger implementation uses the Gini impurity to calculate feature importance. The Gini impurity measures how often a random sample would be mislabelled if it were randomly labelled according to the distribution of labels in the training set. This measure is useful for feature selection as it indicates how much a feature decreases the weighted impurity of a trained decision tree, i.e. how optimally the feature splits the training set. This decrease is averaged across all trees and used to generate a feature importance value for each feature where higher values denote more important features.

Similarly to classifier-independent selection, the feature scores (importances) are ranked from highest to lowest. The indices of the top 25 ranked features are used to generate a 72 element bit array which is saved to a text file. The classification component can later use this file to shorten feature vectors to contain only the 25 most important features before they are used for either training or prediction.

This chapter has reviewed existing point features and feature selection methods, and has described the implementation of the features and methods chosen for the component. These extracted features are used for training and prediction by the classification component, as described in the next chapter. The chapter includes a review of various classifier types and the implementation details of the three classifiers chosen for the framework.

Chapter 5

Point Cloud Classification

A crucial part of the scan campaign classification framework is, of course, the classifier. This chapter starts with a review of existing classifier categories in Section 5.1. Then, the implementation of three classifiers chosen for the second component, Point Classification, is described in Section 5.2 which includes the classifiers' pre-processing, training and prediction phases.

5.1 Related Work

Point cloud classification is an active field of research with many approaches proposed throughout the literature. While their applications vary, these approaches can be sensibly grouped by their classifier type as done in [127]. In the review below, classifiers are therefore split into six categories: instance-based, rule-based, probabilistic, max-margin, ensemble and deep learning. For each section, notable papers are presented and discussed. The review ends with a comparison of the six classifier categories.

Instance-based classifiers

Instance-based classification is different to other classification methods in that no model is learned. Instead, unseen feature vectors are directly compared to known vectors in the "training" set. The similarity between samples is defined by the Euclidean (or other) distance between them; the closer the feature vectors, the more similar the samples. The unseen sample is then assigned the class of the closest training sample.

In [56], the instance-based Nearest Neighbour (NN) algorithm is used to classify points in urban scenes based on structural features extracted from points' spherical neighbourhoods. An extension of the NN algorithm, k-Nearest Neighbour (k-NN), is used in [28].

Rather than assign points the class of the closest point, the paper uses k-NN to classify points as the majority class of its k nearest neighbours. A variety of feature sets were tested with k-NN, including eigen-features and local shape descriptors, and overall accuracy of 92% was achieved on datasets containing cars, poles, trees and walls.

The main advantage of instance-based methods is their obviation of a training phase, which saves time and computational resources. They can also adapt to new knowledge by simply adding or removing instances from the dataset as opposed to re-training a new model. However, their induction phase becomes slower as the dataset and therefore complexity of the prediction hypothesis increases. They can also overfit to noise in the dataset if the closest instance(s) to a test sample is noise.

Rule-based classifiers

Rule-based classifiers learn to represent knowledge as a set of rules, usually in the form of binary decisions. These rules adapt as the classifier encounters new training samples so that they can be applied to unseen samples. A single rule can be likened to an if-statement e.g. if green then grass. A prominent rule-based classifier is the decision tree, where a hierarchy of splits (decisions) are recursively built top-down as new training examples are seen. Some classifiers combine multiple trees for ensemble classification (discussed later in this section).

In [33], a single decision tree is trained on the vertical and horizontal features of vehicle segments in airborne laser scans, enabling vehicle detection that improves on the precision of previous methods by 30%. A decision tree is used in [9] to classify points from urban datasets based on their triangular network and individual features, detecting points belonging to grass and paved areas with accuracy varying from 68% to 92% depending on the wavelength of the laser scanner.

Decision trees are easy to visually interpret and understand when there are a low number of branches or nodes. They can also handle training samples with missing features and are good at handling categorical features. They also require minimal parameter selection and are fast to train. However, while they perform well with small sets of descriptive features they are capable of modelling complex interactions between large feature sets. Decision trees also tend to overfit the training set by learning long decision chains, although there are techniques that attempt to alleviate this by limiting (pruning) the tree's depth.

Probabilistic classifiers

Probabilistic classifiers predict a probability distribution over a set of all possible classes that an unseen sample could belong to. This is different to classifiers that predict a

single class. However, they can predict a sample's class as the one with the highest probability.

The probabilistic Naive Bayesian classifier is trained on segments of 3D models in [62] to detect objects in both indoor and outdoor environments for robotic navigation, outperforming SVMs trained on the same data. Two other probabilistic methods, Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) were both used in [58] to classify tree species based on intensity and structural features, where the QDA achieved up to 99% accuracy on datasets of coniferous and deciduous trees.

The class probabilities provided by these types of methods are advantageous as they can be interpreted as the certainty (or uncertainty) of prediction. Furthermore, due to the probability distribution of related variables these methods can handle missing or noisy data. They are also simpler to implement as their hyper-parameters are learned from the data. However, probabilistic methods rely on assumptions: Naive Bayesian classification assumes that all features are conditionally independent and does not model the relationship between correlated features, while LDA and QDA assume that independent variables are normally distributed. If these assumptions are not met then these methods may be inaccurate.

Max-margin classifiers

Max-margin classifiers maximise the distance between samples from different classes in the feature space. The most commonly used max-margin approach is the SVM, discussed in the background chapter. SVMs maximise the distance between different classes in the feature space by constructing one or more hyperplanes to separate samples. New samples are classified based on which side of the hyperplane they fall into.

In [111], an SVM is used to classify segmented objects in urban point clouds. Candidate objects are identified as discontinuities on the ground and are described by geometrical and contextual features. The authors motivate their choice of SVM by its ability to deal with high-dimensional and limited training sets. Through a hierarchical classification scheme, the method achieved an accuracy of 82% on well-segmented objects in terrestrial and aerial datasets. Another paper [31] uses an SVM to classify regions in urban mobile laser scanner point clouds as either vegetation or non-vegetation. These regions are identified through a multi-resolution clustering, from which different histogram features based on point density or eigenvalues are extracted. Vegetation such as trees and shrubs were classified with 95% accuracy in city center scans when using eigenvalue histograms. SVMs have also been used to classify airborne LIDAR data, as in [140]. Here, segments identified through region-growing are described by 13 geometry, radiometry, topology and echo features. The SVM was shown to achieve greater than 92% average accuracy on three airborne datasets.

SVMs can accurately classify high-dimensional and unstructured data when properly configured. With the right kernel function and regularisation parameters, they can solve complex problems without overfitting to the training data. Using one-versus-one class prediction, they are particularly good at differentiating between similar object classes. However, choosing the best kernel function and parameters is non-trivial and if incorrectly done can weaken the classifier significantly. While some parameters such as the width of the kernel and classification error penalty can be automatically tuned, this adds an overhead to an already computationally complex training phase.

Ensemble classifiers

Ensemble classification involves combining a set of *weak* classifiers into a single *strong* classifier (weak and strong in terms of accuracy). The two most common ensemble approaches are *bagging* and *boosting*. Bagging (bootstrap aggregating) involves training each of the weak classifiers with a randomly drawn subset of the training data and counting their votes to assign a class for prediction. Boosting incrementally builds the ensemble by training newer models with instances that previous models achieved lower accuracy on.

A successful bagging method is the random forest (RF), also described in the background chapter. In [27], an RF is used to classify objects in terrestrial laser scans captured by Google Street View including vehicles and traffic lights. Another paper [141] found RFs effective at classifying points in the KITTI [43] dataset using hybrid image/point cloud features, achieving 89% average accuracy on classes including buildings, pedestrians and vegetation. The authors in [48] use an RF to classify objects in point clouds of cluttered scenes from the NYU-Depth V2 dataset [117], achieving state-of-the-art accuracy at the time. Lastly, in [131] an RF is used to distinguish between tree-like and non-tree-like objects represented by 14 geometric features in mobile laser scans, achieving up to 92% overall accuracy.

One boosting algorithm used for point cloud classification is Adaptive Boosting [40] (AdaBoost), where the ensemble also consists of decision trees. In [126], AdaBoost classifies clusters of points in noisy terrestrial laser scans represented by multi-scale features and achieves more than 90% accuracy on classes including trees, buildings and people. Variations of AdaBoost are used in [71] to classify road, grass, buildings and trees in aerial scans with higher than 92% accuracy using a simple set of 5 LiDAR features. Lastly, AdaBoost is used for face recognition in [30] using Gabor wavelet features extracted from depth and intensity images, achieving a 95% verification rate on the FRGC V2 face recognition dataset [93].

The main strength of ensemble methods comes from their composition of multiple diverse weak learners. By training each learner with different training samples, ensemble

methods have reduced bias and are less prone to overfitting. Random forests go an extra step by randomly sampling the features used to train each tree, further diversifying the ensemble and increasing its robustness. Random forests also provide variable importance measures which can be used for feature selection. One limitation of ensemble methods is their difficulty to visually interpret compared to e.g. a single decision tree. In addition, it takes time to manually choose parameters and choosing poor parameters can decrease accuracy e.g. training too many boosted trees can overfit the training data.

Deep learning

Deep learning methods learn a new representation of training data through multiple (*deep*) levels of transformation. Each level transforms the data into a higher and more abstract level, resulting in complex descriptions of the data. Research involving deep learning for point cloud classification has become very active in recent years. Although they typically involve neural networks, these deep learning methods differ from the *shallow* MLP described in the background chapter, which follows the traditional approach of training on hand-crafted features.

The deep learning network PointNet [96] directly reads point cloud data (XYZ and optional local/global features) as input and learns a set of optimisation functions that select the most descriptive points from the point cloud. During prediction, the output of these functions is aggregated in the final layer to assign point classes. When applied to the ModelNet40 [137] dataset containing 40 synthetic object classes, the network achieved state-of-the-art performance at the time. This network was also used in [63] to classify points represented by a *superpoint graph* that encodes the contextual relationships between points, also achieving state-of-the-art results on both the outdoor Semantic3D [49] and indoor S3DIS [11] datasets. Deep learning was applied to RGB-depth images in [119], where a combination of convolutional neural networks (CNN) and recursive neural networks (RNN) classified depth images of 51 household objects [61] faster and more accurately than comparable architectures.

Deep learning classifiers are highly accurate and can obviate time-consuming feature engineering by learning descriptive representations directly from training data. Furthermore, knowledge from a trained network can be transferred and fine-tuned to new tasks provided the learned representation is suitable. However, supervised deep learning typically requires more labelled data and resources (such as top-end GPUs) than previously discussed classifiers to achieve their full potential. Deep learning also requires important design and parameter choices e.g. the network type, number of hidden layers and learning rates, for which established theory is currently limited due to ongoing advancement in the area. Furthermore, fine-tuning pre-trained networks may not be suitable for datasets or classification tasks that differ greatly from the original problem.

Discussion

It is evident that there are many machine learning methods to choose from for point cloud classification, each with their own advantages and disadvantages. However, it is still difficult to determine how these methods compare to each other when used on large terrestrial laser scans of diverse scenes. To help choose a suitable classifier, Table 5.1 summarises the differences between the six presented classifier types. The speed and accuracy comparisons are approximated from other papers [127][49] where the classifiers were evaluated.

Although simple to implement, instance-based classifiers are not suitable as they have a relatively low accuracy, their prediction is slow and they could overfit the noise found in terrestrial scans. Rule-based methods like single decision trees are very fast but have low accuracy, and attempts at improving accuracy by increasing depth can lead to overfitting. Probabilistic methods have better accuracy, but their assumption of feature independence or the distribution of training data cannot be met reliably with limited training data, as is often the case with campaign-based classification. Max-margin methods are slow but can achieve accuracy with the right parameters. However, these parameters are difficult to choose and tuning them automatically adds even more time to the already long training and prediction stages. Deep learning classification methods achieve the best accuracy given enough labelled data, time and computational power. However, designing deep networks is not trivial or automatable and their demand for large amounts of training data may not be met by manual labelling. A standard *shallow* neural network, such as an MLP trained with engineered features, would require less resources at the cost of accuracy, but may still perform similarly to other discussed classifiers. Lastly, ensemble methods achieve high accuracy while being simpler to implement and tune than maxim-margin and deep learning methods. Random forests, in particular, are fast as they are easily parallelisable and are less prone to overfitting due to their random sampling of samples and features.

After reviewing the literature and comparing existing approaches to point cloud classification, the applicability of three classifiers to campaign-based scan classification stand out: random forests, SVMs and MLPs. These three algorithms have been shown to achieve high accuracy when classifying point clouds. Although they require varying amounts of configuration and have differing training/prediction speeds, they each have their own strengths as discussed in the above review.

	Parameter Setup	Training Speed	Prediction Speed	Generalisability	Accuracy	Example
Instance-based	Simple	No training phase	Slow	Overfit to noise	Medium-low	Nearest Neighbour, k-Nearest Neighbour
Rule-based	Simple	Fast	Very Fast	Overfit if too deep	Low	Decision Tree
Probabilistic	Parameters learned from data	Very Fast	Fast	Assumes independence or normality	Medium	Naive Bayesian, LDA, QDA
Max-margin	Non-trivial, slow if automated	Slow	Very Slow	High with good regularisation parameter	High	SVM
Ensemble	Low difficulty, automation feasible	Fast (RF) Slow (AB)	Moderate	Diversification of samples/ features reduces bias	High	Random Forest AdaBoost
Deep learning	Non-trivial, automation infeasible	Slow	Very Fast	Overfit if too many hidden layers	Very High	CNN, RNN, MLP

TABLE 5.1: Summary of reviewed classifier types' parameter setup, training and prediction speeds, generalisability, accuracy and examples.

5.2 Implementation

The classification component handles all of the training and prediction in the framework. It use features extracted by the features component paired with point labels to learn a model which can predict the class of unseen points. This section explains the component's implementation of necessary pre-processing steps, three different classifiers and unseen point prediction.

5.2.1 Pre-processing

Labels

Before training or prediction, the necessary point labels are loaded and processed. In the case of training these are the labels used to teach the classifier the mapping between feature vectors and classes, and in the case of prediction these are the ground truth labels used to evaluate the accuracy of the classifier. Since the format and encoding of these labels can vary between datasets, they are restructured to be consistent and easily read by the component. Specifically, labels are saved to a plain text file where the line number of each label corresponds to the index of its respective point in the cloud. Labels are integers between 0 and the total number of classes, while unlabelled points are given a label of -1.

Scans

Raw scans with varying formats are initially read to `pcl::PointCloud` objects before they are downsampled to a resolution of 2.5cm and re-written to disk as `.pcd` (Point Cloud Data) files using PCL's `io::savePCDFileASCII` method. Like label processing, this is done to ensure consistent behaviour within the component. During downsampling, a new label file is also generated to store the labels of the downsampled point cloud. This is done by assigning downsampled points the dominant label of the points that were averaged to generate them. The `.pcd` point cloud is then treated as the base point cloud for all subsequent training and prediction. Although these point clouds have lower resolution and detail than the original point clouds, they provide enough detail to successfully compare the methods within the framework while requiring much less time to process.

Class Balancing

The last pre-processing step is balancing the class samples before they are used for training. This is needed since most terrestrial scans have large class imbalances e.g. many more points of buildings than vegetation. If the classifier learns from an imbalanced

training set it could develop a bias towards the over-represented class and perform inaccurately on under-represented classes. The implementation first oversamples smaller classes by duplicating random samples until the class contains n samples, then oversamples larger classes by replacing them with n samples selected randomly without replacement. This results in the same number of samples per class and a balanced training set. Note that this is done directly on label data before any feature extraction occurs.

5.2.2 Training

Three different classifiers were implemented for use in the component. The choice of these classifiers is based on the review and discussion of methods in the literature presented in Section 5.1. They represent three algorithms identified as applicable to campaign-based classification: SVMs, MLPs and random forests. This section provides the implementation details and parameter choices of each classifier.

Regardless of which classifier is to be trained, the component loads the indices of the previously balanced training points and the scans they belong to. Features belonging to the points at these indices are then extracted and the resulting vectors are reduced according to the stored 72 element bit array if feature selection is enabled. An additional feature normalisation step (Equation 5.1) is applied to the training features where x is the original feature, x' is the normalised feature and $\overline{x}$ and σ are the mean and standard deviation of the feature. This step is necessary due to the importance of magnitude in some of the classifiers' objective functions and ensures that features are equally weighted when training begins. Once prepared, the feature vectors and their respective class labels are are passed to the classifier.

$$x' = \frac{x - \overline{x}}{\sigma} \tag{5.1}$$

SVM implementation

The SVM is implemented using the OpenCV [16] C++ library's SVM class which is based on the LibSVM [18] library. Although OpenCV is primarily aimed towards computer vision tasks, it also contains a number of standard machine learning algorithms. LibSVM is an established and actively supported SVM library which provides the required SVM functionality for the component.

The SVM uses c-support vector classification. This is one of the two available n-class ($n \geq 2$) classification modes in LibSVM, the other being v-support vector classification. These modes have similar functionality and performance but use different regularisation

parameters (c or v) to penalise misclassifications while training. The choice mostly depends on which parameter the user finds easier to interpret. The radial basis function (RBF) kernel is used to map training samples into higher-dimensional space as described in Section 2.3.2. This is the default kernel function in LibSVM and is commonly used throughout the literature.

The SVM is trained using the OpenCV's SVM `trainAuto` method. The method is set to use 5-fold cross validation grid-search to find the optimal cost parameter c and the RBF kernel's γ parameter from the library's default grid of parameters. Although this adds to training time, it is a worthwhile trade-off as poorly chosen parameters reduce the accuracy of the SVM. Once the SVM model is trained it is saved as a `.yml` file to be loaded before prediction.

MLP implementation

The MLP is built using the neural network module in OpenCV. Although there are other highly configurable and powerful libraries capable of training deep networks such as Caffe [55] and Tensorflow [8], the OpenCV module provides the necessary methods to train a standard MLP. As discussed previously, a standard *shallow* MLP is more feasible for training on a limited amount data than deeper networks, and is more straightforward to design. The lighter OpenCV implementation is therefore sufficient for the purposes of this research, and yields a classifier more comparable to the implemented SVM and random forest.

The architecture and parameters of the component's MLP were chosen based on various preliminary tests and guidelines in the literature. The input layer has the standard $n\_features$ neurons and the output layer has $n\_classes$ neurons. The hidden layer contains the mean number of neurons in the input and output layer, i.e. $(n\_features + n\_classes)/2$ neurons. The network uses a symmetrical sigmoid activation function to compute neuron outputs and the resilient back-propagation (RPROP) algorithm [101] to update the weights of the network. The RPROP algorithm is chosen for its flexibility as it uses only the signs of gradients to update weights while the symmetrical sigmoid function is the most commonly used method for modelling non-linear relationships in the training data. This typical pairing of methods is used throughout the literature and provides a MLP that fairly represents the algorithm.

Before training, the order of training samples is randomly shuffled - this reduces variance and the likelihood that the MLP overfits the data. The network is then trained until either 10000 training iterations are completed or until the difference in error between iterations reaches $\epsilon = 0.0001$ (1e-4). Once trained, the metadata and weights of the network are written to a `.yml` file to be loaded at the start of the prediction phase.

Random Forest implementation

The Ranger C++ library [136] is used for its random forest implementation of the original Breiman algorithm [17]. Although the OpenCV machine learning module offers a random forest algorithm, it was not used for the component as it did not support the parallel training of trees. In addition to providing parallel training of trees, the Ranger library is actively supported and configurable.

There are a number of important parameters to set when training the random forest. Their values were determined via basic trial-and-error testing, the implementation of a parameter tuning phase as well as reference to the literature. A forest of 100 trees with a target partition size of 1 is trained. This is the minimum partition size and allows trees to be fully grown. Although letting a single decision tree grow to maximum depth can result in overfitting, this is alleviated by the reduction in variance caused by training 100 trees on different parts of the training set. Tree nodes are split with $\sqrt{p}$ (rounded down) randomly selected features , where p is the size of the feature vector. With $p = 72$ features, nodes are therefore split with 8 randomly selected features. The Gini index is chosen instead of information gain as a splitting criterion since they yield similar accuracies while computing the information gain is slower due to its logarithmic function. Lastly, the number of processing threads is set to the maximum available threads determined via the standard C++ `thread::hardware_concurrency` method. This significantly reduces training times by dedicating a separate thread to each tree being trained.

An additional method (`Data::loadFromVectors`) was added to support the loading of feature vectors from memory as only reading from `.dat` files was supported initially. This was done to improve the pipeline's speed and standardise the interfacing between the feature extraction and classification components. Once the training vectors are loaded and the random forest's parameters are set the random forest is trained. After all 100 trees are trained, the forest and its metadata are written to a `.forest` file to be loaded when the prediction phase begins.

A summary of the methods and parameters used for each of the three implemented classifiers is given in Table 5.2.

Support Vector Machine	
Library	OpenCV (LibSVM)
Vector classification	c-support
Kernel	Radial basis function (RBF)
Parameter tuning (c and γ)	Grid search
Cross validation folds	5
Multi-Layer Perceptron	
Library	OpenCV
Input layer size	$n\_features = 72$
Output layer size	$n\_classes$
Hidden layer size	$(n\_features + n\_classes)/2$
Activation function	Symmetrical sigmoid
Training method	Resilient back-propagation (Rprop)
Random Forest	
Library	Ranger
Trees	100
Target partition size	1
Split features	$\lfloor\sqrt{n\_features}\rfloor = \lfloor\sqrt{72}\rfloor = 8$
Splitting criterion	Gini index
Threads	`thread::hardware_concurrency` = 6

TABLE 5.2: Important methods and parameters used by the implemented SVM, MLP and RF.

5.2.3 Prediction

The implementation of the prediction phase is straightforward. The classifier (either SVM, MLP or RF) is initialised and loads the trained model from the saved `.yml` or `.forest` file.

Before they are classified, feature vectors of the test points are processed if necessary. If the classifier was trained on a subset of selected features, the feature vectors are reduced to contain only the selection according to the stored feature indices. The test feature vectors are then normalised using the saved mean $\overline{x}$ and standard deviation σ of each feature.

Rather than immediately classifying a test vector as it is extracted, vectors are stored and predicted in batches of 100,000. This improves the efficiency of the pipeline by limiting the number of function calls substantially without consuming an unsafe amount of

memory. The prediction phase is also accelerated by the parallel extraction of test feature vectors using the OpenMP [25] library. Specifically, a separate thread is dedicated to extract a point's features at each resolution scale (2.5cm, 5cm and 10cm) before they are combined into a single vector.

As each batch of test vectors reaches its maximum size, the classifier iterates through them to predict point classes. For each point cloud being classified, a new cloud with points coloured according to their class is generated and saved as a `.pcd`. If ground truth labels are available, an additional *error cloud* that highlights misclassifications and a `.csv` file containing the true and predicted classes of each point are generated. These files are later used to generate statistics and evaluate the classifier.

This chapter reviewed classifiers in the literature and detailed the implementation of three classifiers chosen for the point classification component. Combined with the previous chapter's features component, these form the basis of the framework and enable the investigation of feature sets and classifiers applied to terrestrial laser scans. The next chapter describes the final component, which identifies representative scans that the first two components to use for training.

Chapter 6

Representative Scan Selection

An accurate scan campaign classifier requires a descriptive set of training scans to learn from. The third and final component of the framework, Representative Scan Selection, automatically selects descriptive scans for the other components to extract features from and train with. Given that such a method does not currently exist, the chapter starts with a short review of loosely related but informative methods in Section 6.1. The design and implementation of the proposed method's three *schemes* are then described in Sections 6.2 and 6.3.

6.1 Related Work

An early survey of the literature did not reveal previous work on the selection of representative campaign scans for labelling and training. However, existing methods like *instance selection* and *active learning* may help when designing a representative scan selection component. The goal of these methods — to identify a descriptive subset of data — could make them informative despite some core differences. This review categorises and describes methods of instance selection and active learning, using examples from previous surveys [84][112], and discusses their applicability to representative scan selection.

6.1.1 Instance Selection

Instance selection methods reduce the size of a labelled training set T. The goal is to find a subset $S \subseteq T$ such that a classifier trained on T achieves similar accuracy when trained on S. When successful, instance selection reduces the time and memory needed for training while maintaining (or improving) classification accuracy. The motivation is

therefore similar to feature selection described in Section 4.1.2 of the Point Features and Feature Selection chapter.

Instance selection can be separated into two types: filter and wrapper methods. Examples of these methods and their their applicability to scan selection discussed below.

Filter Methods

Filter methods select $S \subseteq T$ without the use of a classifier. Instead, instances are evaluated and selected based on their properties and relationship with other instances. Notable examples of filter methods are discussed below.

Nearest Neighbour Search

Some filter methods use nearest neighbour search to determine *border* and *interior* instances. A border instance has a different class to its nearest neighbour, whereas an interior instance belongs to the same class as their nearest neighbour. Due to border instances' ability to differentiate between classes, methods like Pattern by Ordered Projections (POP) [102] and Pair Opposite Class-Nearest Neighbour (POC-NN) [99] discard interior instances and select instances that border the most classes.

Clustering Methods

Clustering is also used for instances selection, where T is split into n clusters and the center of each cluster [72] or the center of merged same-class clusters [76] are selected. One clustering method, Object Selection by Cluster (OSC) [82], selects interior and border instances from homogeneous (same class) and inhomogeneous clusters. Here, an instance is a border if its nearest neighbour in the same cluster has a different class, and the center of homogeneous clusters are also selected to represent uniform regions.

Weighting Methods

Weighting methods assign scores to instances which are used to threshold and select instances. For example, Weighting Prototypes (WP) [86] computes an instance's weight based on their nearest neighbours and *enemies* (instances belonging to a different class), and discards instances with weights above a threshold. Another method, Prototype Selection by Relevance (PSR) [83], selects a percentage of relevant instances i.e. instances with the highest average similarity to their class, as well as the most similar instances belonging to different classes.

Wrapper Methods

Wrapper methods are classifier-dependent; they select a subset S of labelled instances based on the accuracy a classifier achieves when they are used for training. Examples of prominent wrapper methods are presented below.

Nearest Neighbour Classifier Methods

Many wrapper methods use a k-Nearest Neighbour (k-NN) classifier to evaluate and select instances. An early k-NN method, Condensed Nearest Neighbour (CNN) [53], classifies T with a k-NN classifier trained on randomly selected instances from each class, then adds misclassified instances to S. A more recent variant, Generalized CNN (GCNN) [21], only selects instances that are not *absorbed* (represented) by S, which improves efficiency and accuracy. Another k-NN method and its variants, Decremental Reduced Optimization Procedure (DROP1-5) [135], involves discarding an instance if it is not needed to correctly classify its *associates* (instances whose nearest neighbours include the instance under consideration).

Support Vector Machine Methods

An SVM essentially performs instance selection when it identifies instances to use as support vectors, as these vectors discriminate between classes. To utilise this, one method [68] applies the DROP2 algorithm to the support vectors generated by an SVM. Another method, Support Vector k-Nearest Neighbour Clustering (SV-kNNC) [120] clusters support vectors with k-means, then selects instances from homogeneous clusters and majority-class instances from inhomogeneous clusters.

Sequential Search

Sequential search-based methods iteratively add or remove instances from S based on their contribution towards a classifier's accuracy. For example, the Backward Sequential Edition (BSE) [81] method discards instances until accuracy begins to diminish, while the Sequential Floating Search (SFS) [95] method allows discarded instances to be re-added if they improve accuracy. Although search methods are expensive due to the number of steps and classifier re-training, they can be used with any classifier.

Discussion

Unfortunately, instance selection methods do not solve representative scan selection. Filter methods require labelled training data, and wrapper methods require both labels and the training of a classifier. As such, these methods cannot be used directly for representative scan selection which occurs *before* labelling or training. However, they include methods that could work with unlabelled data, such as those that cluster instances (OSC, SV-kNNC) or select them by their similarity (PSR). As clustering methods are traditionally unsupervised and similarity functions work without labels, these methods may help identify representative scans for labelling.

6.1.2 Active Learning

Active learning proposes that a classifier's accuracy can improve if given the ability to choose which instances are labelled for training. It is particularly useful when there is an abundance of unlabelled data but labels are difficult to obtain. The classifier starts by training on a small set of training data and then *querying* an *oracle* (e.g. an expert user) to label informative examples. In doing this, only the most descriptive instance are labelled and the accuracy of the classifier is increased.

Instances are chosen for labelling based on a *querying strategy*. A number of strategies, differing by how they evaluate instances, are proposed throughout the literature. Six notable querying strategies are presented below followed by a discussion of their relevance to scan selection.

Uncertainty Sampling
Uncertainty sampling [67] involves querying (requesting labels for) instances that the classifier is the least certain about i.e. instances with the lowest probability of being correctly classified. This can be measured simply by the classifier's confidence in its prediction, or by the difference between the classifier's two most likely predictions (margin sampling). Another approach, entropy sampling, considers the confidence of all classes when measuring uncertainty.

Query-by-committee
Query-by-committee sampling [115] utilises a group or *committee* of classifiers trained on the same labelled instances to predict (vote for) the class of unlabelled instances. The oracle is asked to label the instances that the committee disagrees on the most i.e. the instances with the most differing votes. Although slow, any type and number of classifiers can be used, and there are different ways of measure the committee's disagreement e.g. *soft-voting* where the classifiers' confidence in all classes are used rather a single vote.

Expected Model Change
The expected model change strategy asks the oracle to label instances that would cause the most change to the classifier if its class was known. One method, Expected Gradient Length (EGL) [114], applies this strategy to any classifier that uses gradient descent e.g. linear SVMs or neural networks. Specifically, it queries instances that would increase the magnitude of the training gradient (vector) the most if they were added to the training set.

Expected Error Reduction
The expected error reduction strategy [103] queries instances that are expected to reduce the classifier's generalisation error the most, which should reduces the classifier's total

number of incorrect predictions. The strategy is particularly expensive as it requires re-training the classifier for each possible query, as well as estimating the expected future error of the classifier on the rest of the unlabelled instances.

Variance Reduction

Variance reduction is a response to how expensive the error reduction strategy is. It uses the idea that a classifier's expected error can be split into three terms [44]: noise, bias and *variance*. By reducing variance, expected error is therefore indirectly reduced. This is less expensive than error reduction as it does not require the re-training of classifiers, but is still slower than most strategies for larger feature vectors.

Density Weighted Models

Many strategies tend to query high-impact border instances that change the classifier in some way. While these are good for differentiation, they do not represent common or interior instances. Strategies like error and variance reduction indirectly avoid this problem by considering all unlabelled instances, but this is expensive. Density weighted models [113] ensure that interior instances are queried by weighting their scores (computed by e.g. uncertainty sampling) by their average *similarity* to other unlabelled instances. One density approach [80] clusters instances to identify representative instances and avoid outliers using a local noise model.

Discussion

As with instance selection, active learning is not directly applicable to representative scan selection as queries require a trained classifier. However, there are some transferable ideas. For example, density weighted models query instances that are similar to others, with one approach using clustering to identify representative instances. As with instance selection, these similarity and clustering methods do not require labelled data and could therefore help select unlabelled scans for training. Furthermore, the idea of multiple strategies that target different traits or effects on the dataset could be adopted by the component to determine important training scan properties.

6.2 Design

Developing the scan selection component required a different approach to the other components. The feature and classifier components are based on widely understood methods chosen to test their applicability to various terrestrial laser scanning campaigns. However, the scan selection component has little literature to draw on since the problem of automatically selecting a representative set of scans from a scanning campaign is not currently addressed.

In order to address the problem, a set of selection criteria were developed. The aim of these criteria is that they, when satisfied, yield a descriptive set of scans for labelling and training. A method to automatically select scans based on these criteria was then developed. Parts of the method borrow techniques from instance selection and active learning which were explored in the related work, Section 6.1.

The remainder of this section is split into two parts. First, the selection criteria are presented and explained. A high-level description and motivation of the component's design is then given. A detailed explanation of the methods used to realise this design is given later in the implementation Section, 6.3.

6.2.1 Selection Criteria

Before designing the selection component, four selection criteria were developed. They are based on practical experience of choosing scans for labelling and discussions with scan labelling practitioners. The four criteria are explained below:

1. **Scans should contain classes commonly found in other scans**. If the dataset contains scans with many points from trees and buildings, the scans selected for labelling must have enough of these points so that the classifier can learn to recognise them in unseen scans.

2. **Scans should not be too similar to each other**. Due to the way that the scanner is incrementally moved around the site, there is the potential for very similar scans in the dataset. While this is useful when generating a 3D model of an area of interest, too many similar scans diminish the quality of a training set as they add little new data for the classifier to learn from.

3. **Scans should contain a diverse array of classes**. In some scans, almost all points belong to a single class e.g. a close-up scan of a wall. While such scans can be useful if the points belong to a class underrepresented in the dataset, more

diverse scans should be selected as they provide more classes and boundaries to learn from.

4. **Scans should represent all scene types at least once**. There are some scenes that do not occur often in a dataset but are still important to label. For example, in a dataset of mostly outdoor scans a few indoor scans may be recorded. Despite the indoor scans forming a small part of the dataset, the classifier needs to learn from at least one of them in order to adequately classify the others.

6.2.2 Component Design

The design of the component is separated into two parts: **Representation** and **Selection**. Representation generates two feature vectors for each candidate scan which are then used by one of three *selection schemes* to identify scans based on the selection criteria. Parts of the component's design, particularly the use of clustering methods and similarity measures, are inspired by their usage in the instance selection and active learning literature (reviewed in Sections 6.1.1 and 6.1.2).

Representation

After considering the selection criteria, it is evident that a method choosing scans based on their criteria compliance would first require knowledge of certain scan properties. For example, in order to evaluate how diverse a scan's point classes are (for criteria #3), there needs to be a measure of the frequency or distribution of points belonging to each class in the scan. With labelled scans this is trivial - a scan's class distribution can be directly computed from labels. However, a sensible way to make such measurements on unlabelled scans is not obvious.

The proposed design therefore uses *clustering* in lieu of existing labels as a way of estimating the distribution (and other properties) of points' classes in unlabelled scans. For example, if all the points in a scan campaign were clustered, the ID of each point's cluster could be viewed as its class label, and a distribution could be calculated.

Unfortunately, this is too costly due to the millions (or billions) of point feature vectors that would require extraction before clustering. The design therefore opts to cluster *segments* resulting from an over-segmentation phase (using the VCCS algorithm introduced in Section 2.2.1). This way, the cluster assignments of a scan's segments (not points) may be used to estimate the scan's point distribution and other properties. It is important to note that *all* the segments from *all* the scans in the dataset are clustered at once, rather than clustering the segments of each scan separately.

This approach allows feature vectors to be computed for each segment rather than each point in the dataset. A succinct set of 12 features is chosen to reduce the dimensionality of the feature space and preserve meaningful distance metrics when clustering. No neighbourhood search is required for these features; instead, the supervoxel segment itself is treated as the point neighbourhood and each of the 12 features is computed only once per supervoxel. This is much faster than finding each point's k-nearest neighbourhood, extracting individual point features and clustering them. A summary of the proposed supervoxel clustering approach is illustrated in Figure 6.1.

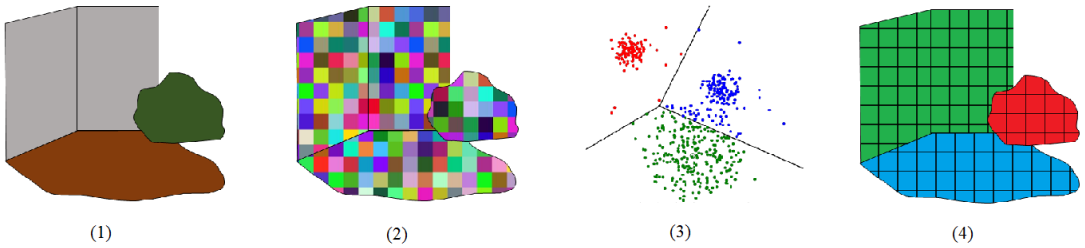


FIGURE 6.1: Supervoxel clustering pipeline: **(1)** A raw, unlabelled scan with walls (grey), ground (brown) and shrub (green) points. **(2)** Scan after VCCS over-segmentation into supervoxels coloured by their unique IDs. **(3)** Visualisation of k-means clustering of supervoxel feature vectors. **(4)** Scan with supervoxels "labelled" with the ID of the cluster they were assigned to.

To cluster the segments' feature vectors, the k-means algorithm (described in Section 2.3.5) is used. Although k-means is unable to find non-linearly separable clusters like more precise clustering methods (e.g. DBScan), it is fast and requires only one parameter. The accuracy of the k-means clustering step (and the preceding segmentation step) greatly impact how well the cluster assignments can describe scans. However, even rough properties derived from segments' cluster assignments should provide enough information about scans for them to be evaluated against the selection criteria.

Two properties are derived from the cluster assignments: *distribution* and *similarity*. Distribution measures the spread of cluster assignments in a scan and is described by a scan's description vector D. Figure 6.2 illustrates the distribution of a scan's supervoxel cluster assignments - in this case the distribution vector D would comprise of 3 elements (one for each cluster) such that $D = [0.34, 0.51, 0.15]$.

The similarity vector S measures how similar a scan's segments are to the clusters they were assigned to. To illustrate this, Figure 6.3 visualises a basic clustering of supervoxels. In this figure, a scan's similarity score for a cluster would be calculated as the average *cosine similarity* between its supervoxels (solid points) and the center of the cluster they were assigned to (points outlined black).

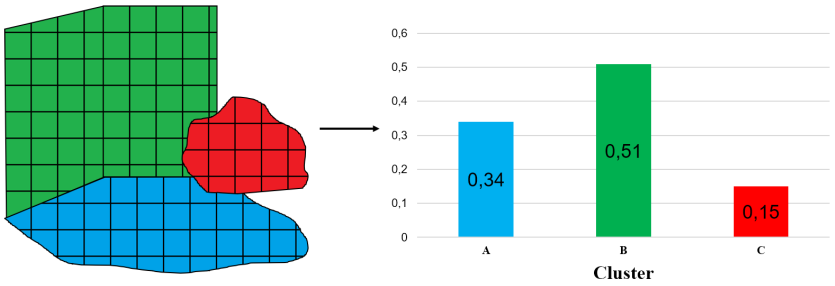


FIGURE 6.2: Distribution of a scan's supervoxel cluster assignments across three clusters A, B and C. This scan's distribution vector $D = [0.34, 0.51, 0.15]$

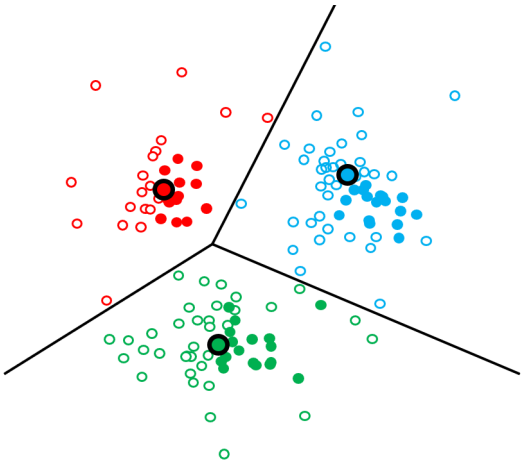


FIGURE 6.3: Example k-means clustering visualisation containing: supervoxels belonging to a scan (solid points), supervoxels belonging to other scans in the dataset (hollow points) and the center of each supervoxel cluster as computed by k-means (black-outlined points).

Vectors D and S therefore approximate the distribution of points in a scan and how similar a scan's points are to other points in the campaign. Although only estimates, these vectors describe important characteristics which can be used to differentiate scans. Using the vectors alone, the second part of the component evaluates how well a scan complies with one or more of the selection criteria in order to identify representative scans for labelling.

Selection

The second part of the component is designed to identify a descriptive set of scans using their distribution and similarity feature vectors D and S generated in the previous step. Scans are evaluated by how well their distribution and similarity features meet one or more of the selection criteria and the best performing scans are chosen for labelling.

Rather than focusing on a single method of evaluating scans, three separate scan selection *schemes* were developed. These schemes differ in the importance they assign each

of the four selection criteria. The motivation behind this is that by comparing the classification performance of the scans chosen by each scheme, the relative importance of the selection criteria may be revealed. It also avoids the scenario of a single overly-complex "silver bullet" method unsuccessfully trying to meet all the criteria at once. Due to the exploratory nature of the component it is more sensible to design three different schemes and gauge their performance separately.

Each selection scheme chooses N (determined by the user) scans to be labelled. A high-level explanation of the three schemes and the criteria they target is given below including illustrative figures. Although each scheme targets specific criteria, they are not necessarily mutually exclusive. More detail on the implementation of these schemes is given later in the implementation Section, 6.3.

1. **Balanced scheme.** This scheme is designed to select the most balanced scans in the dataset. It uses the previously estimated distribution vector D to determine how balanced each scan is. If all the segments in a scan belong to one cluster then it is less balanced than a scan with segments belonging to multiple clusters. To illustrate this, an example of a well-balanced scan is presented in Figure 6.4, where the proportion of supervoxels belonging to each cluster are roughly the same at 0.33. Scans are given a score based on how much their distributions differ from a hypothetical, perfectly balanced (uniform) scan's distribution. The N best (lowest) scoring scans are selected for labelling. This scheme primarily targets criteria **#3**: "Scans should contain a diverse array of classes".

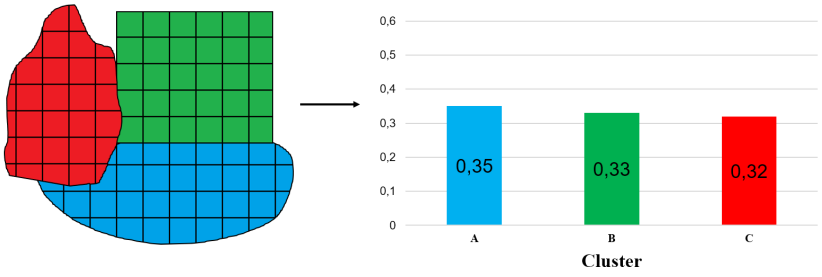


FIGURE 6.4: Example of a well-balanced scan distribution; supervoxels belong to clusters almost equally with three proportions ≈ 0.33.

2. **Similarity scheme.** The similarity scheme favours scans with segments commonly found throughout the campaign. It leverages the scan's cluster similarity vector S alone. For example, if a scan's segments have a high cosine similarity to the other segments in their clusters, this indicates that the scan's points represent point classes commonly found throughout the campaign. To illustrate this, Figure 6.5 shows how close the supervoxels (represented by solid points) of two scans, **a** and **b**, are to the center of their clusters (represented by black-outlined points).

In this figure, it is likely that scan **a** has more supervoxels similar to the rest of the campaign than scan **b** since its supervoxels are closer to the centers of the clusters they belong to. A score is given to each scan based on its supervoxels' average cosine similarities to their centers and the N best (highest) scoring scans are chosen for labelling. This scheme focuses on criteria **#1**: "Scans should contain classes commonly found in other scans".

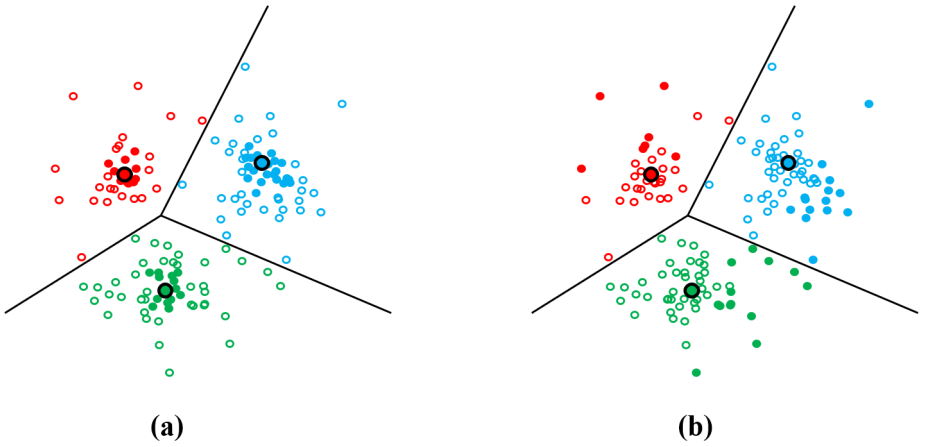


FIGURE 6.5: Supervoxel cluster assignments of two scans: **(a)** a scan's supervoxels (solid points) near the center (black-outlined points) of the clusters they were assigned to, **(b)** a second scan's supervoxels further away from their cluster centers.

3. **Distinct scheme.** This scheme selects a distinct set of scans which maximally vary from each other. It combines both the distribution vector D and similarity vector S to form a feature vector V for each scan. The scans themselves (not their segments) are then clustered by their V vectors into N (number of scans to be selected) groups. Since clusters are inherently dissimilar to each other, the scan closest to the center of each cluster is chosen to form a distinct set of N scans for labelling. This scheme is illustrated in Figure 6.6, where 35 scans are clustered into $k{=}N{=}5$ groups, and the 5 scans closest to their cluster centers in the V feature space are selected for labelling. This scheme aims to meet criteria **#2**: "Scans should not be too similar to each other" in addition to criteria **#4**: "Scans should represent all scene types at least once".

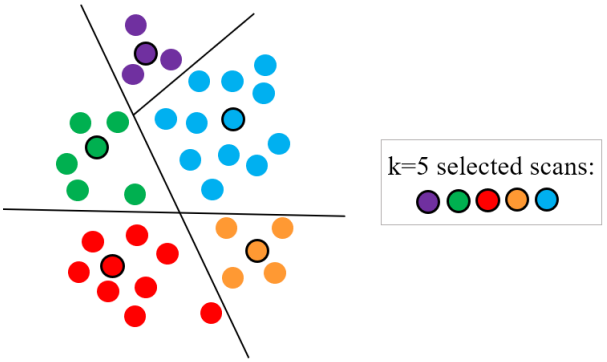


FIGURE 6.6: k-means clustering of 36 scans in the V feature space with k=5, scans closest to their cluster center outlined black.

6.3 Implementation

This section describes the implementation details of the design proposed in the previous section and is also split into two steps: Representation and Selection.

6.3.1 Step 1: Representation

The first step is to generate a representation of every scan in the campaign that can be used the three schemes to select scans. This step uses a pipeline of segmentation, feature extraction and clustering methods to generate the distribution vector D and similarity vector S for each scan. A description of each stage of the pipeline is given below.

Segmentation

Scans are segmented into supervoxels using PCL's VCCS implementation available in the `SupervoxelClustering` class. Important parameters for this method are voxel resolution R_voxel and seed resolution R_seed, where R_seed is the resolution of the grid that the starting seeds of supervoxels are evenly distributed across, and R_voxel is the resolution of the octree used to maintain the adjacency graph that generates the supervoxels. It was found that ideal values for these parameters (yielding the best segmentation) depend on the point density of the scans. They are typically set to $R\_voxel \approx 0.025\text{m}$ and $0.25\text{m} \geqslant R\_seed \leqslant 0.5\text{m}$ for the terrestrial laser scans used in this research. Other important parameters are the colour, spatial and normal weights (w_colour, $w\_spatial$ and w_norm) set between 0 and 1 that, by weighting the equation used for voxel distances, determine the effect that these properties have on supervoxel generation. These parameters are set at w_colour=0, $w\_spatial$=0.3 and w_norm=1.0 which yields slightly irregularly shaped segments that ignore (unreliable) colour values

but strongly respect normal orientation e.g separate adjacent segments on the wall and the ground.

Feature extraction

A total of 12 features are extracted from each supervoxel. The first 10 features, derived from the supervoxel's height data and local structure matrix's eigenvalues, are chosen based on their consistently high performance in previous studies [128]. Specifically, these features are: verticality, linearity, planarity, surface variation, sphericity, anisotropy, height difference, height deviation as well as the sum and ratio of 2D eigenvalues. A formal definition of these features was given earlier in the feature chapter's Table 4.1. The `voxelSize` and `centroidNormZ` properties of the supervoxel are also used as they are fast to extract (already computed during segmentation) and describe the supervoxel well. Specifically, `voxelSize` is the number of points in the supervoxel and `centroidNormZ` is the Z component of the normal to the supervoxel surface at the centroid point. These two features help to further differentiate supervoxels e.g. a supervoxel on a tree trunk is typically smaller than, and perpendicular to, a supervoxel on the ground.

Clustering

The mlpack library's [24] `kmeans` method is used to group supervoxels from all the scans into k clusters using their normalised feature vectors. The method's k parameter, which defines the number of clusters to be found, is determined using the so-called *elbow method*. This method computes the total sum of squared errors within each cluster (i.e intra-cluster variance) for k= 2...10 and plots them on a line graph, as shown in Figure 6.7.

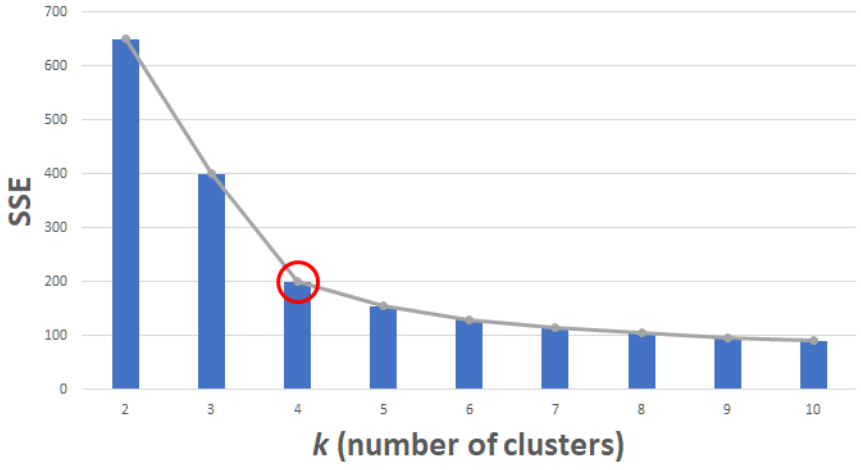


FIGURE 6.7: Elbow method graph with optimal $k = 4$.

The point in the line graph where increasing k begins to yield a diminishing reduction in variance denotes the optimal k. This point in the graph, indicated by the red circle in Figure 6.7, resembles an elbow and is determined by the user. Although this method

requires user input, identifying the elbow point is intuitive and is required only once per dataset.

Distribution and Similarity vectors

After the supervoxels are clustered into the optimal k number of groups, the resulting assignments are used to calculate a distribution vector D and similarity vector S for each scan.

The distribution vector D describes the spread of supervoxels in a scan, and comprises separate scores for each cluster. A scan's distribution score for a cluster is simply the proportion of its supervoxels belonging to the cluster. This is computed as $P_i = N_i/T$, where P_i is the distribution score for cluster i, N_i is the number of supervoxels in the scan belonging to cluster i and T is the total number of supervoxels in the scan. Dividing by T normalises scores to [0,1] which ensures that differently sized scans with similar distributions are sensibly compared by the selection schemes in Step 2. Finally, a scan's distribution scores are combined to form its distribution vector $D = [P_1, P_2, ..., P_k]$, where P_i is the proportion of the scan's segments belonging to cluster i and k is the total number of clusters.

The similarity vector uses the cosine similarity function to describe how similar a scan's supervoxels are to their clusters. The cosine similarity function defined in Equation 6.1 measures the similarity between two vectors A and B based on the angle between them - the smaller the angle, the greater the similarity. This cosine similarity function has previously been used to measure intra-cluster similarity as well as the similarity between the centroids of two clusters [121]. It conveniently yields a value between -1 (completely dissimilar) and +1 (completely similar).

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|B\|} \tag{6.1}$$

A scan's similarity score for a cluster C is calculated as the average cosine similarity between its supervoxels in C and the center of C. This is formalised in Equation 6.2, where C_i is the cosine similarity score for cluster i, $SV_{i,j}$ is the jth supervoxel in cluster i, Cen_i is the center of cluster i and N_i is the total number of supervoxels in cluster i.

$$C_i = \frac{\sum_{j=1}^{N_i} \text{sim}(SV_{i,j}, Cen_i)}{N_i} \tag{6.2}$$

A scan's cosine similarity scores for each cluster are then combined to form its similarity vector $S = [C_1, C_2, ..., C_k]$, where $C_i \in [-1, 1]$ is the average cosine similarity between

the scan's segments in cluster i and their center, and k is the total number of clusters. Since the center of the cluster represents the mean supervoxel in the cluster, this is a reasonable way of comparing a supervoxel to its cluster and is less expensive than comparing it to all other cluster members. The sum of supervoxel-center similarities are averaged to ensure that they fall into the [-1:1] range - this way the scores are indifferent to cluster size and only reflect similarity.

6.3.2 Step 2: Selection

The selection step uses the previously generated distribution vector D or similarity vector S (or both) to choose a final subset of N scans to be labelled. Three different scan selection schemes were implemented. The implementation of each scheme is explained below.

1. Balanced scheme
This scheme selects the most balanced scans in the dataset i.e. scans with segments most evenly assigned to the clusters. To implement this scheme, only the scan's cluster distribution vector $D = [P_1, P_2, ..., P_k]$ is required, where P_i is the proportion of the scan's segments belonging to cluster i and k is the total number of clusters. The scan's balance goal $G = 1/k$ is also calculated - this is the hypothetical value of P_i in a perfectly balanced scan. A `balanceScore` $= |(P_1 - G)| + |(P_2 - G)| + ... + |(P_k - G)|$ is calculated for each scan. The N scans with the lowest `balanceScore`, i.e. the scores that differ the least from the perfect score of 0, are selected for labelling.

2. Similarity scheme
This scheme favours scans with segments commonly found throughout the dataset i.e. scans with segments most similar to their clusters. The scheme requires the scans' previously generated cosine similarity vectors $S = [C_1, C_2, ..., C_k]$ where $C_i \in [-1, 1]$ is the average cosine similarity between the scan's segments in cluster i and their center, and k is the total number of clusters. A `similarityScore` $= (C_1 + C_2 + ... + C_k)$ is summed for each scan. Since a cosine similarity of -1 and +1 denote a perfect dissimilarity and similarity respectively, the N scans with the highest `similarityScore` are selected for labelling.

3. Distinct scheme
The scheme aims to select a distinct set of scans i.e. scans that differ from each other as much as possible. To do this, each scan's distribution vector D and similarity vector S are concatenated to form a new feature vector $V = [D_1, D_2, ..., D_n, S_1, S_2, ..., S_n]$ where D_i and S_i are the ith elements of vectors D and S and n is the (equal) size of the vectors. These feature vectors are then stored as rows of a matrix and normalised column-wise

using Equation 5.1 used previously for point features. The scan feature vectors are then clustered using the mlpack `kmeans` method, with $k = N$ (the total number of scans to select) and the `AllowEmptyClusters` flag set to false. This flag forces the k-means algorithm to cluster the scans into N groups. Although k is not optimally set e.g. with the elbow method (as done when clustering supervoxels), the k-means algorithm still attempts to maximise the difference of the clusters. The scheme does not assign a score to each scan, instead it chooses a representative scan from each cluster - the scan closest to the cluster center. This is simply the scan with the smallest Euclidean distance $d = \sqrt{(V_1 - CV_1)^2 + (V_2 - CV_2)^2 + ... + (V_n - CV_n)^2}$, where V is the scan's feature vector and CV is feature vector of the center of the cluster the scan was assigned to. The scan with the smallest distance d is chosen from each cluster, resulting in a selection of N distinct scans for labelling.

Once a scheme has selected N scans their indices are stored for the classifier to load before training. In the case that there are no labels for the scans, the user can refer to the system output for the list of scans to label.

This chapter has presented the design and implementation of the representative scan selection component, following a brief review of potentially informative methods. This component, comprising three schemes based on various selection criteria, enables the investigation of whether scans automatically identified as representative can yield a more accurate classifier than randomly selected scans. As the details of all three components have now been established, the next chapter presents the experiments conducted on the components and analyses the results.

Chapter 7

Evaluation

This chapter describes the experiments used to test the scan campaign classification framework and analyses the results. It is split into an experiments section and a results section. The experiments section introduces the test datasets, system specifications, performance metrics and testing process. The results section then presents and analyses the results relevant to each of the framework's components.

7.1 Experiments

Before describing the experiments, it is important to recall the aims of the research. The first aim is to investigate the performance of various classifiers, features and feature selection methods when used to classify large and diverse terrestrial laser scanning campaign scans. The second research aim is to determine if an automated selection of descriptive campaign scans can outperform a random selection when labelled and used for classifier training. To meet these aims, three core components - classification, features and scan selection - comprised of competing methods and schemes were implemented. Only through rigorous testing can these methods be evaluated and the outcome of the research be determined.

The experiments are therefore designed to measure the performance of each component's methods when used as part of a scan classification pipeline. Specifically, the experiments use relevant metrics to measure the unseen scan classification speed and accuracy of different classifiers, feature selection methods and scan selection schemes. The remainder of this section describes the test datasets, system specifications, performance metrics and the testing process used to test each method.

7.1.1 Test Datasets and System

Rather than testing the methods on a single campaign, four different datasets were used for variation and to enable a more comprehensive analysis of the methods. These datasets contain scans from (mostly) outdoor scenes varying by size and the number of classes they contain. Datasets with existing labels were chosen as manually labelling enough scans would have prolonged testing. Two of the datasets are from cultural heritage preservation campaigns and two are from general terrestrial laser scanning campaigns.

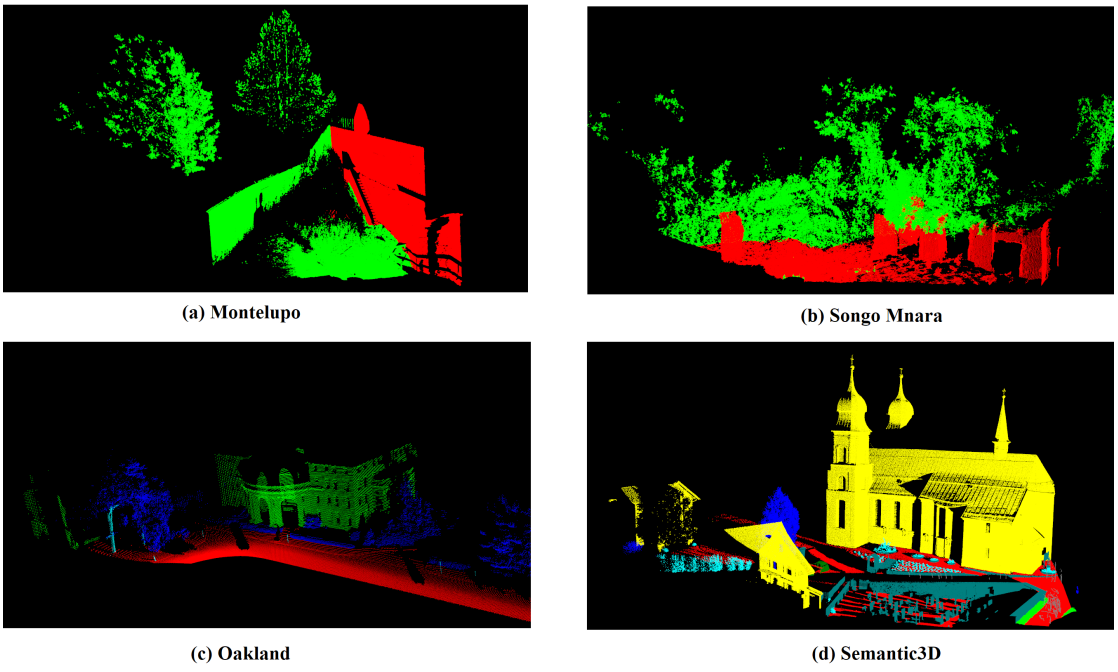


FIGURE 7.1: Example ground truth scans from the **(a)** Montelupo [7], **(b)** Songo Mnara [5], **(c)** Oakland [78] and **(d)** Semantic3D [49] datasets. The binary datasets **(a,b)** are labelled *Keep* (red) and *Discard* (green), while datasets **(c,d)** have multiple class labels.

All of the test datasets' scans were downsampled to a point density of 2.5cm, i.e. at most one point per 2.5cm^3, using the octree method described in Section 4.2.1. This baseline downsampling is intended to improve the consistency of each method's performance across datasets. Downsampling also accelerates prediction speeds which, depending on the original density and number of points in a dataset, can be very slow. Although downsampling reduces the level of detail in scans, it does not invalidate the experiments since the methods are tested on the same data. In addition to downsampling, some of the datasets' scans were split into two or four "scans" to increase the number of candidate scans for the scan selection component. This was done by splitting scans by the sign of their points' X coordinates (two halves) or by the sign of their X *and* Y coordinates

(four quadrants). Example ground truth scans from each dataset are shown in Figures 7.1.a - 7.1.d. The datasets are described below and summarised in Table 7.1.

Songo Mnara: This scanning campaign, provided by the Zamani project [5] at UCT, surveys the ruins of a Tanzanian town with crumbling buildings surrounded by trees and plants. The 10 original scans of the dataset were split in half yielding 20 scans. There are only two classes: *Keep* and *Discard*. Points within an *area of interest* (the cultural heritage site) belong to the keep class while the vegetation inside and the points outside this area belong to the discard class.

Montelupo: This cultural heritage campaign, provided by the VCG lab [7] at CNR-ISTI, contains 8 scans split into 16 and covers an old Italian church inside a walled area with many plants. Most of the scans survey the exterior walls but some are taken from inside the church. Like Songo Mnara, points are labelled as *Keep* or *Discard*. Points from the area of interest (the church) are kept while vegetation around the area is discarded.

Oakland 3D: The Oakland 3D set [78] of 15 point clouds covers a university campus in Oakland, USA. There are five classes: building, vegetation, ground, pole and wire. These scans are from an urban scene with organised and regularly shaped structures, which provide contrast to the cluttered and degraded objects in the previous two datasets. Furthermore, the dataset's particularly low density (only 1.3M points across 15 scans) could help identify the more robust methods.

Semantic3D: The original Semantic3D training set [49] contains 15 high resolution point clouds of sites recorded across Switzerland and Austria. Of these, five scans containing a church, fountain and various structures were split into quadrants yielding 20 point clouds. There are eight classes including building, high vegetation, low vegetation and natural terrain. This dataset contains the most points and classes of the four, and allows for intensive testing.

Name	**Description**	**# Scans**	**# Points**	**Classes**
Songo Mnara	Ruins of town in Tanzania	20	8.8M	2: keep, discard
Montelupo	Old church in Italy	16	6.4M	2: keep, discard
Oakland	University campus in Oakland, USA	15	1.3M	5: building, vegetation, ground, pole, wire
Semantic3D	Various structures in Switzerland and Austria	20	41.6M	8: man-made terrain, natural terrain, high vegetation, low vegetation, building, hardscape, scanning artefact, car

TABLE 7.1: Summary of test datasets including description, number of scans, points and classes.

For the binary datasets, 3000 samples per class are used for training, while 1000 training samples are used for each class in the multiclass datasets. This sums to a total of 6000 (Songo Mnara), 6000 (Montelupo), 8000 (Semantic3D) and 5000 (Oakland) training samples. Although classification benchmarks exist for the Semantic3D and Oakland datasets, the framework is only tested on a subset of these datasets' training data that has been re-purposed to evaluate the framework's methods.

Test system specifications
Tests were run on a PC with a hex-core Intel Core i5 CPU clocked at 4.4 GHz and 16GB of DDR4 memory running Ubuntu 16.04. The test datasets are stored on a 6Gb/s solid-state drive which greatly accelerates I/O, but does not influence speed measurements as they only start once the point clouds have been loaded into memory.

Performance metrics
A collection of metrics were used to measure the speed and accuracy of the methods. These metrics were chosen based on their common usage throughout the literature. The same set of measurements are generated for each test run, of which the most relevant and insightful are presented in the results Section 7.2. A definition of these metrics and how they were implemented is given below.

7.1.2 Quantitative Metrics

Speed: The C++ `std::chrono` library was used for all timing measurements. It was used to track the duration of feature extraction, classifier training and point prediction. Only operations belonging to these tasks are tracked; the times for surrounding processes e.g. disk I/O are ignored.

Accuracy: The scikit-learn [90] library's `sklearn.metrics` module was used for all quantitative accuracy measurements. A Python script was created to load a `.csv` file containing the true and predicted classes of each point in the test set and generate the following:

- The precision, recall and F1 score of each class. To understand these, let TP_C and TN_C be the true positives and true negatives of a class C, while FP_C and FN_C are the false positives and false negatives of C. Figure 7.2 illustrates the relationship between these values. Precision is the fraction of points classified as C that are relevant (actually belong to C) i.e. $\frac{TP_C}{TP_C+FP_C}$, while recall is the fraction of relevant points that are correctly classified i.e. $\frac{TP_C}{TP_C+FN_C}$. In essence, high precision means that a significant number of points classified as C actually belong to C, while high recall means that a significant number of points belonging to C were correctly classified. Due

to the trade-off between these two metrics, an F1 score combines them by calculating their harmonic mean: $\text{F1} = 2 * \frac{precision*recall}{precision+recall}$. A high F1 score signifies that the classifier has both high precision and recall.

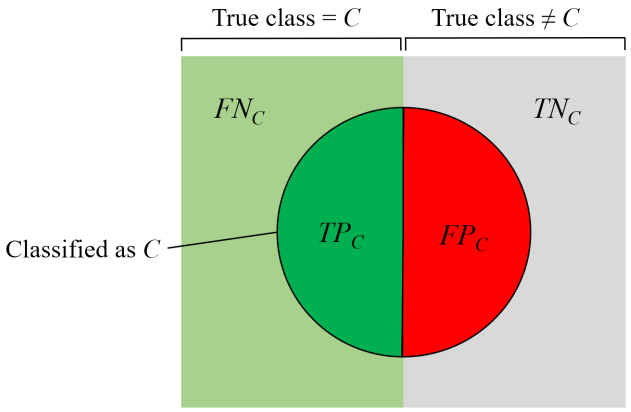


FIGURE 7.2: Illustration of true/false positives/negatives. The circle denotes points classified as C, while the left and right rectangles denote points that do and *do not* truly belong to C respectively.

- The micro, macro and weighted averages of the class precision, recall and F1 scores. The micro-average of a metric is calculated by first summing individual class contributions to form TP_{all}, TN_{all},FP_{all} and FN_{all} and then re-computing the metric that is consequently influenced by any class imbalances. The macro-average of a metric is simply the sum of the classes' independently calculated metrics divided by the total number of classes. Although this is faster to compute, it does not account for class imbalances. The weighted-average addresses this by weighting independent class metrics by the proportion of their examples in the dataset before they are averaged.

- The overall accuracy of the classifier. This is the fraction of points in the test dataset that are correctly classified. Note that this is effectively the same as the micro-average precision, i.e. $\frac{TP_{all}}{TP_{all}+FP_{all}}$, but is presented separately due to its intuitive meaning.

- A confusion matrix that visualises the classifier's performance. This is a table where each row represents the points in a true class and each column represents the points in a predicted class, such that the diagonal shows the true positives for each class. Figure 7.3 illustrates this with two example matrices, **a** and **b**, where **a** uses raw counts and **b** normalises counts by their class sizes. The layout of these matrices helps identify which classes a classifier "confuses" with others.

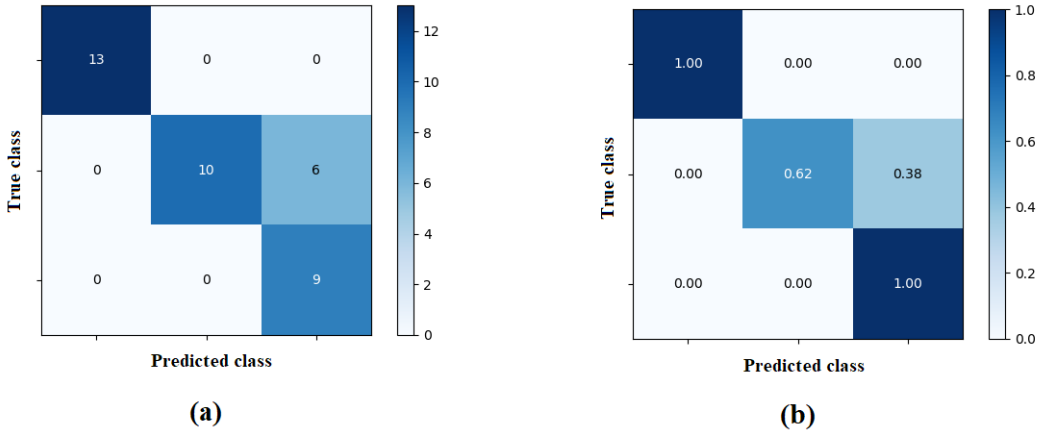


FIGURE 7.3: Example confusion matrices. Diagonals show true positives - points with the same predicted and true class. Matrix **(a)** contains raw counts, matrix **(b)** normalises counts by class sizes. Adapted from scikit-learn website[1].

In addition to the above, one of two possible measurements are stored if the classifier was trained on features selected by a feature selection method. These are the aggregated *feature ranks* that were calculated if the classifier-independent filter selection method was used, or the *feature importances* that were generated if the RF-dependent feature importance method was used.

7.1.3 Qualitative Metrics

In order to qualitatively evaluate the accuracy of a classifier's predictions, a *prediction cloud* and an *error cloud* are generated for each classified test scan using PCL's `io::savePCDFileASCII` method. The prediction cloud simply colours points according to their predicted classes, giving an overall impression of the classifier's performance. For more insight, the error cloud colours points by their predicted classes but displays incorrectly classified points a high contrast colour. Examples of these point clouds are given in Figure 7.4, where **a** shows a ground truth scan with 3 classes, **b** shows a classifier's attempt at classifying the ground truth and **c** highlights the incorrectly classified points. These visualisations help reveal the classifier's most challenging areas or classes.

[1]Confusion matrix tutorial available at: https://scikit-learn.org/0.17/auto_examples/model_selection/plot_confusion_matrix.html

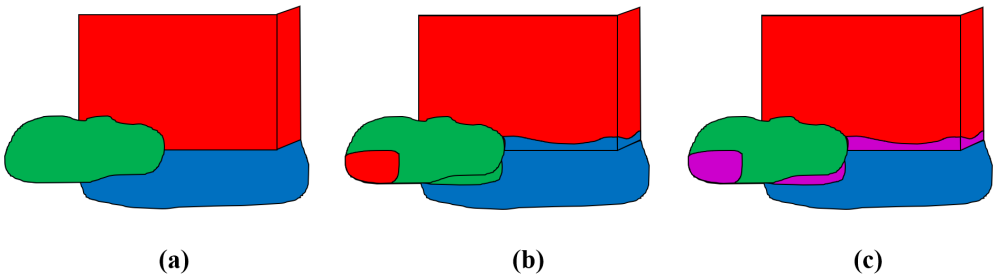


FIGURE 7.4: Example point cloud visualisations. **(a)** A unseen ground truth scan with three classes. **(b)** A classifier's prediction of the points in the unseen scan. **(c)** Point cloud with the classifier's incorrect predictions highlighted.

7.1.4 Testing Process

The testing process was separated into two steps. The first step tested classifiers and feature selection methods simultaneously. The second step then used the outcome of the first step to test the scan selection schemes.

Datasets are split into training and prediction scans depending on the methods being tested and the size of the dataset. Specifically, tests during step one (classifiers and feature selection) use the same set of training and prediction scans for each dataset to ensure that the methods are compared with equal conditions. However, when testing scan selection, training scans are selected (or randomised) and therefore vary between runs. In this case, the prediction set consists of scans that were *not* selected.

Step 1: Classifiers and feature selection methods

Three classifiers and two feature selection methods were tested. The classifiers are the SVM, MLP and RF, the implementations and parameters of which were detailed in Section 5.2. The feature selection methods tested were the filter-based (classifier-independent) method and the RF-importance (classifier-dependent) method described in Section 4.2 of the chapter on features. The set of all extracted features is also tested.

Each of the three classifiers are tested with each of the three feature selections, resulting in nine sets of results (the previously discussed metrics) per dataset. This way, a classifier may be evaluated using its average performance when paired with the feature selection methods, and vice-versa. An example of how these results may look is given in Table 7.2, where the averages of the rows and columns reveal that "Classifier 3" and "Feature Set 1" yield the highest average overall accuracy.

This approach is not exhaustive; only one configuration of each algorithm's parameters is tested and only one run is conducted for each pair of classifier and feature set. This is because test runs are already slow, and adding algorithm variants or more runs would

	Feature Set 1	**Feature Set 2**	**Feature Set 3**	*Avg*
Classifier 1	0.7	0.66	0.71	0.69
Classifier 2	0.66	0.67	0.63	0.65
Classifier 3	0.78	0.72	0.74	**0.75**
Avg	**0.71**	0.68	0.69	

TABLE 7.2: Table of (hypothetical) overall accuracy results of different classifier and feature set combinations, with highest averages in bold.

cause a combinatorial effect and greatly increase test times. Instead, the methods are tested on four separate datasets. Since there are nine classifier-feature selection combinations, this step yields 36 (9×4) separate sets of results for in-depth analysis. To further mitigate randomness and ensure reliability, the methods are tested on identical sets of training samples.

Step 2: Scan selection schemes

The second step tests the three designed scan selection schemes. These are the *balanced*, *similarity* and *distinct* schemes as detailed in Section 6.3 of the scan selection chapter. This occurs after the previous step as only the best-performing classifier and feature set combination is used to test the schemes. This consistent baseline ensures that only the effect of the selection schemes is captured.

A total of ten runs are made per dataset: one for each scheme and seven where scans are selected randomly. This step therefore yields 40 (10×4) sets of results. Multiple random runs are required since the results of different random selections are expected to vary. By doing this, the minimum, maximum and average performance of the random runs can be compared to the selection schemes' performance. Note that the selection schemes are deterministic and therefore do not require multiple runs; they always make the same selection.

7.2 Results

This section presents the results of the experiments described above, and is split into three sections: 7.2.1 Features, 7.2.2 Classifiers and 7.2.3 Scan Selection. Through the analysis of numerous tables, graphs and classified scans, this section provides the information needed to answer the questions of this research.

7.2.1 Features

Feature extraction speed

The feature extraction times for each dataset are shown in Table 7.3. The results show that the average time to extract feature vectors from 1000 prediction samples is between 17ms and 31ms, depending on the dataset. This means that 72,000 features (72 features per sample) are extracted in < 0.1s. Although difficult to compare due to different test sets and system specifications, these times are close to a similar approach [50] where 12M vectors are extracted in 191s (average 16ms per 1000 samples), and much faster than another approach [132] where feature extraction of the same 12M points took 23,000s (average 2s per 1000 samples).

	Training			**Prediction**		
	# of points	**Feature extraction**	**Average (1k points)**	**# of points**	**Feature extraction**	**Average (1k points)**
Montelupo	6000	0.20s	34ms	4,783,495	150.55s	31ms
Songo Mnara	6000	0.13s	22ms	6,354,173	112.08s	18ms
Oakland	5000	0.08s	16ms	872,593	15.07s	17ms
Semantic3D	8000	0.18s	22ms	9,070,732	190.37s	21ms

TABLE 7.3: Number of samples and feature extraction times for each dataset during training and prediction.

Three main factors help accelerate feature extraction times. First, OpenMP allows the parallel feature extraction of points from point clouds with different resolutions. In addition, C++ compiler optimisation flags (e.g. -O2) greatly improve the speed of the Eigen library's matrix calculations. Lastly, using a small radius (5cm) for cylindrical neighbourhoods is much faster than larger radii (e.g. 50cm), especially on datasets with many vertical structures. For example, feature extraction takes longer on Montelupo than Songo Mnara, despite Montelupo having fewer points, due to the large number of walls it contains.

There are no separate extraction times for the feature sets chosen by feature selection. This is because selected sets are likely to still require eigendecomposition or cylindrical

neighbourhood search, the most expensive parts of extraction. It is therefore more practical to extract all features and reduce them than it is to extract selected features only. A more important observation - the speed of using selected features for training and prediction - is analysed later in the classifier results, Section 7.2.2 .

Comparison of Features

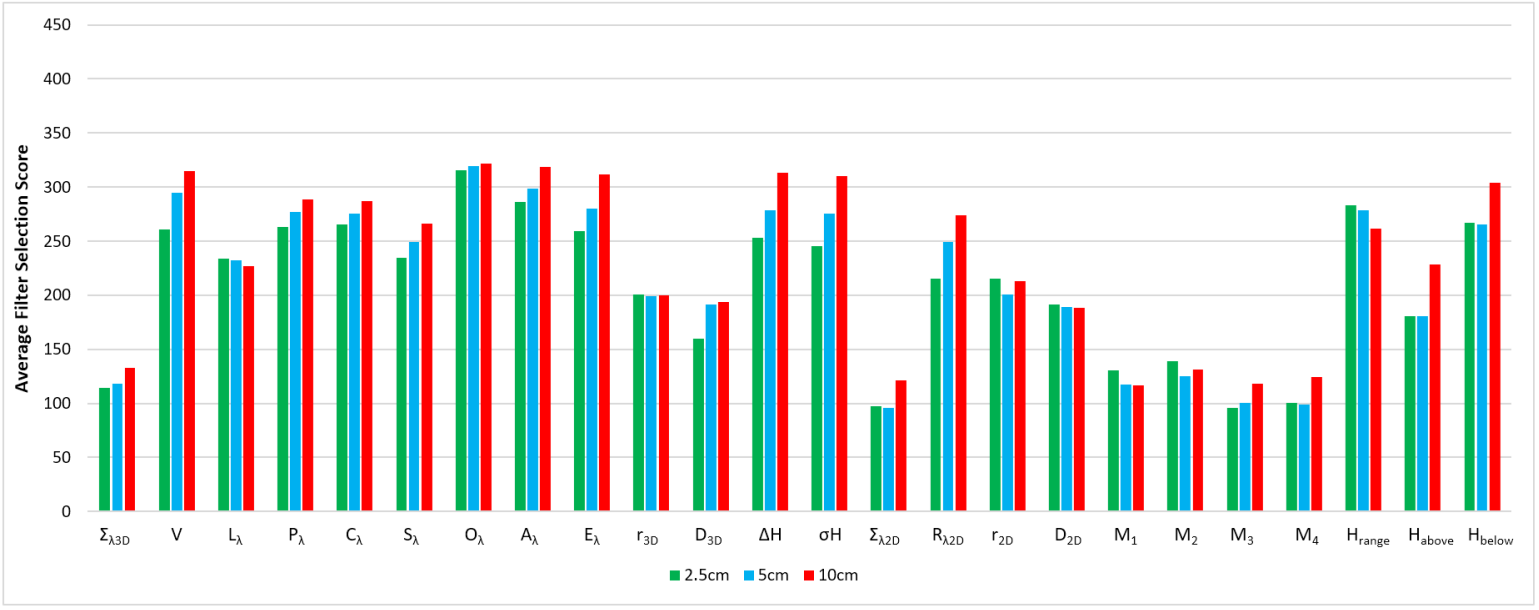


FIGURE 7.5: Average filter selection score of each feature across all four datasets, grouped by feature and coloured by resolution.

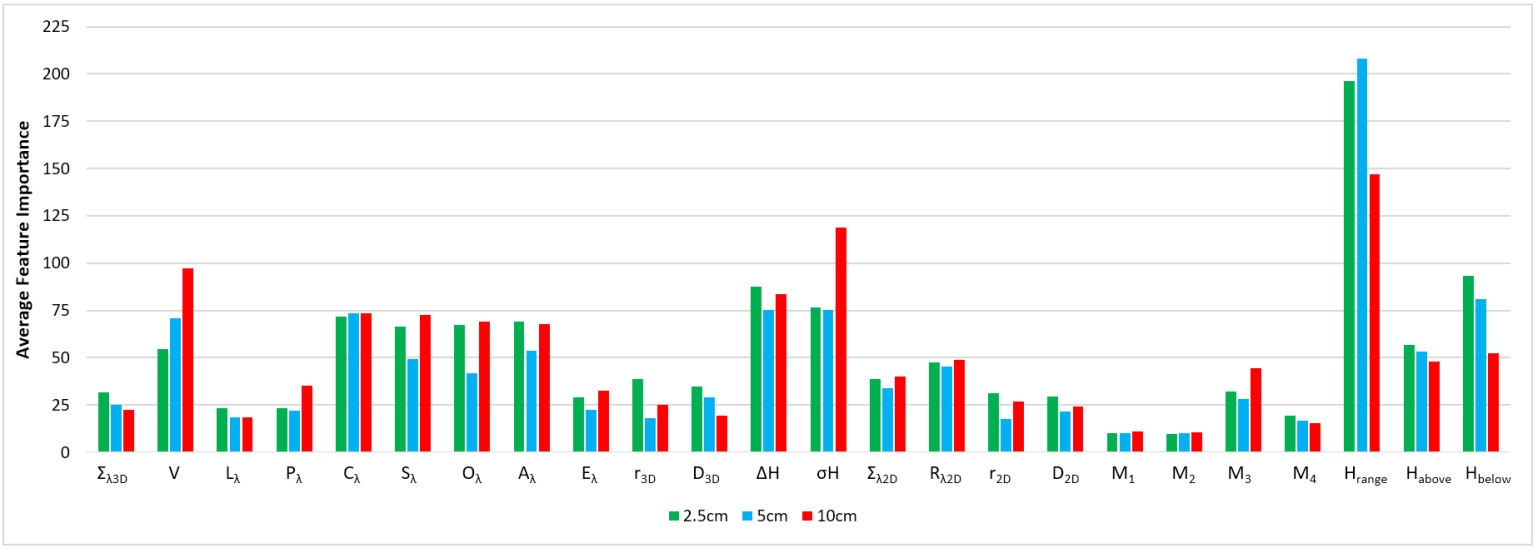


FIGURE 7.6: Average feature importance of each feature across all four datasets, grouped by feature and coloured by feature resolution.

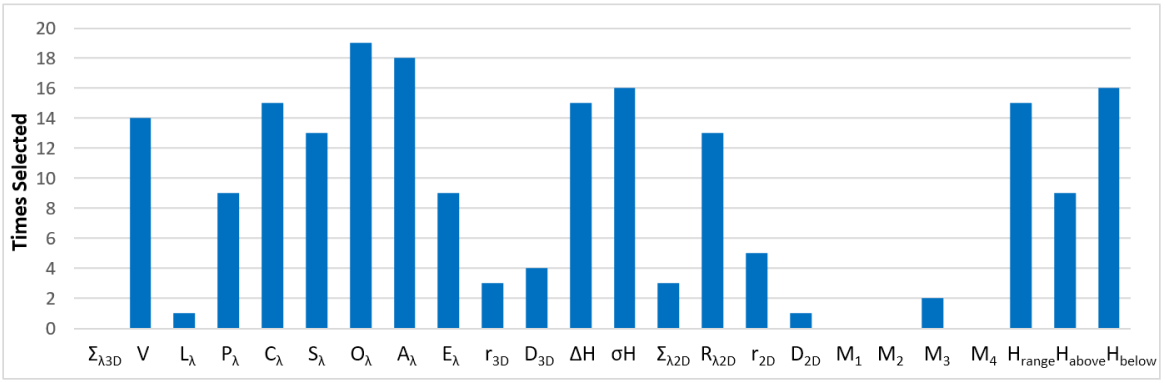


FIGURE 7.7: Number of times each feature (at any resolution) is selected by filter or importance selection on all four datasets.

Features are evaluated using the scores assigned to them by each feature selection method and by the number of times they were selected. Figure 7.5 shows the average filter selection score of each feature, while Figure 7.6 shows the average feature importance score. These are the average scores across all four datasets and are colour-coded by the resolution of the point cloud the feature was extracted from. In addition to these, Figure 7.7 shows the number of times a feature was selected by either method, regardless of resolution.

The results show that some features scored well on both selection methods at one or more resolution: V (verticality), C_λ (curvature), O_λ (omnivariance), A_λ (anisotropy), ΔH (height difference), σH (height deviation), H_{range} (vertical range) and H_{below} (height below). Other features score low on both selections: L_λ (linearity), $\Sigma_{\lambda 3D}$ (sum of 3D eigenvalues), $\Sigma_{\lambda 2D}$ (sum of 2D eigenvalues), M_1 - M_4 (moments), D_{3D} (3D density) and D_{2D} (2D density). These findings are consistent (where applicable) with a previous feature ranking study [130] on the Oakland dataset.

Some features achieved high scores with one selection method but low scores on the other. Filter-based selection scored all (except $\Sigma_{\lambda 3D}$) 3D shape features highly, but importance selection scored L_λ (linearity), P_λ (planarity) and E_λ (eigenentropy) amongst the lowest. Importance-selection highly favoured height features like H_{range} which achieved double the score of the next best feature, whereas filter-selection scored height features well but relatively close to others.

The multi-scale approach of extracting features at 2.5cm, 5cm and 10cm resolutions had a limited impact on feature scores. Lower resolutions are somewhat favoured, with 10cm, 5cm and 2.5cm features being selected 78, 63 and 59 times respectively. However, features like V and σH do much better at 10cm, while H_{range} does better at 2.5cm/5cm. This is likely because V and σH benefit more from k-nearest neighbour search's increased context at lower resolutions, whereas H_{range} is extracted from a cylindrical (r=5cm)

neighbourhood that could miss important points at lower resolutions. If the frameworks implementation of multi-scale features had more levels (as in another approach [50]) or larger intervals e.g. 5cm, 20cm, 50cm, there may have been a greater difference between features scores at different resolutions.

Comparison of Feature Selection Methods

a) Montelupo

	All	Filtered	Important
RF	**0.8822**	0.8732	0.8817
SVM	**0.8614**	0.8479	0.8421
MLP	0.8370	**0.8553**	0.8404
Average	**0.8602**	0.8588	0.8547

b) Songo Mnara

	All	Filtered	Important
RF	**0.9135**	0.9028	0.9094
SVM	**0.9010**	0.8181	0.8904
MLP	0.8807	**0.8960**	0.8902
Average	**0.8984**	0.8723	0.8967

c) Oakland

	All	Filtered	Important
RF	**0.8853**	0.8812	0.8794
SVM	0.8672	**0.8788**	0.8661
MLP	**0.8754**	0.8726	0.8678
Average	0.8760	**0.8775**	0.8711

d) Semantic3D

	All	Filtered	Important
RF	**0.7725**	0.72301	0.7606
SVM	0.7379	0.7082	**0.7478**
MLP	0.7072	0.6999	**0.7181**
Average	0.7392	0.7103	**0.7422**

TABLE 7.4: Overall accuracy of each classifier/feature set combination for each dataset. Highest scoring feature set per classifier in bold.

	All	Filtered	Important
Montelupo	**0.8602**	0.8588	0.8547
Songo Mnara	**0.8984**	0.8723	0.8967
Oakland	0.8760	**0.8775**	0.8711
Semantic3D	0.7392	0.7103	**0.7422**
Average	**0.8434**	0.8297	0.8412

TABLE 7.5: Summary of each feature set's overall accuracy on each dataset. Highest overall accuracy per dataset in bold.

To evaluate the impact of using selected features instead of all features, Table 7.4 gives the overall accuracy of each classifier and feature set combination for each dataset. The averages of these scores is then summarised in Table 7.5. These results show an almost negligible difference in accuracy when using selected features over all features. Indeed, the difference in average accuracy between all and filter-selected features is 1.4%, while the difference between all and importance-selected features is 0.2%, as seen in Table 7.5. In some cases this difference is larger, e.g. on Semantic3D (RF) all features achieved roughly 5% higher accuracy than filter-selected features, and on Songo Mnara (SVM) the

classification results achieved when using all features were around 8% more accurate than when using filter-selected features. Overall, these results show no significant difference in accuracy between all and selected features.

This is a desirable effect of feature selection; using fewer features (resources) with minimal cost to accuracy. However, the implemented selection methods had favourable conditions. Although 25 features are selected (roughly a third of all 72 features), this is a lot considering that there are only 24 unique features extracted at 3 different resolutions. Furthermore, the minor increment in resolution (2.5cm, 5cm, 10cm) may have created redundant features, meaning there are fewer useful features for a selection method to "miss". If the original set consisted of 72 unique features, or if only 10 features were selected by the methods, the selected sets may have achieved much lower accuracy than the original set.

Regardless of the minor accuracy difference, there are some noteworthy interactions between classifiers and feature sets. While the RF was always most accurate when using all features, the MLP was most accurate on 3 of 4 datasets when using a smaller feature set. This may be due to the RF's large number of max-depth decision trees successfully mapping a large input vector, whereas the MLP's shallow design (few hidden layers and nodes) performs its best with fewer inputs. The RF was also more accurate when using important features over filtered features on 3 of 4 datasets - this is expected since importance-based selection is dependent on a trained RF. Lastly, despite filter-selection being classifier-independent, it did not consistently outperform importance selection on the non-RF classifiers (SVM and MLP) as may be expected — on Semantic3D, important features were 4% (SVM) and 2% (MLP) more accurate than filter-selected features.

Overall, the trade-off between accuracy and resources saved is worth it under the right conditions. Depending on the quality of the original set, the size of the reduced set and the type of classifier used, the advantages of feature selection will vary. It is particularly useful for classifiers with training and prediction times that scale poorly with large feature sets, as is discussed later in the classifier results, Section (7.2.2).

Highlighted example: Semantic3D

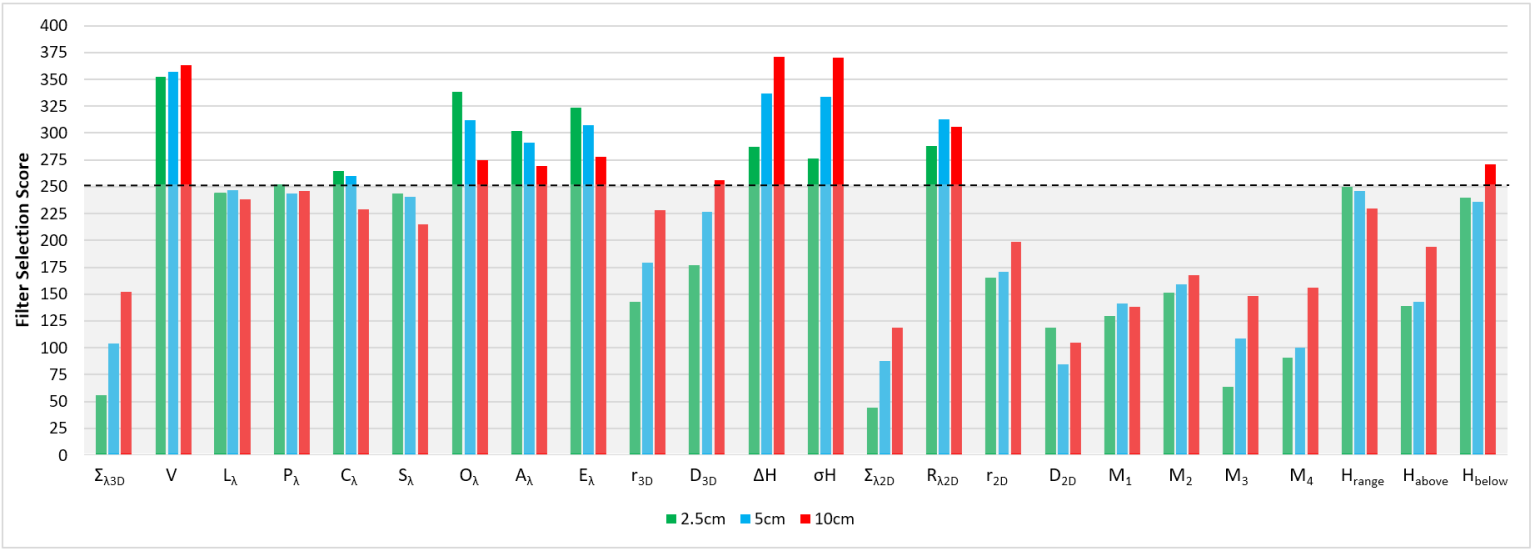


FIGURE 7.8: Filter selection scores of features extracted from the Semantic3D dataset, grouped by feature and coloured by resolution. The 25 top-scoring features (above dashed line) are selected for training.

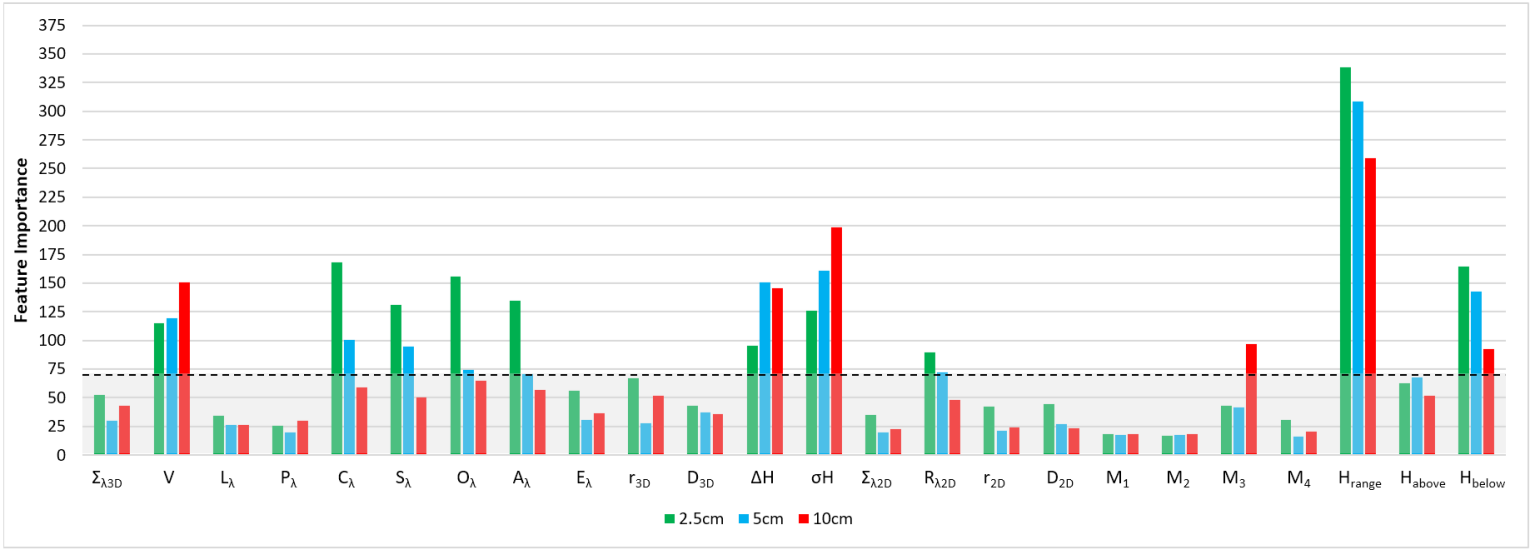


FIGURE 7.9: Feature importance of features extracted from the Semantic3D dataset, grouped by feature and coloured by resolution. The 25 top-scoring features (above dashed line) are selected for training.

This highlighted example illustrates and explains one of the larger differences in accuracy between the feature selection methods. On the Semantic3D dataset, the RF trained on important features achieved 4% higher overall accuracy than the filter-selected features, as was shown in Table 7.4. This is relatively large difference compared to the other results and is worth investigating.

The feature scores given by each method are shown in Figure 7.8 (filter selection) and Figure 7.9 (importance selection). In these graphs, the 25 top-scoring features (those above the dashed line) are chosen by the respective feature selection method. The results show that the majority of selected features are shared by each method at one or more resolution. Of the features not selected by both methods, H_{range} (vertical range) stands out; it has the largest feature importance score by a large margin, but is just missed by the filter-selection method. Based on these scores, the difference in accuracy between the two feature sets may be explained by whether or not they include H_{range}.

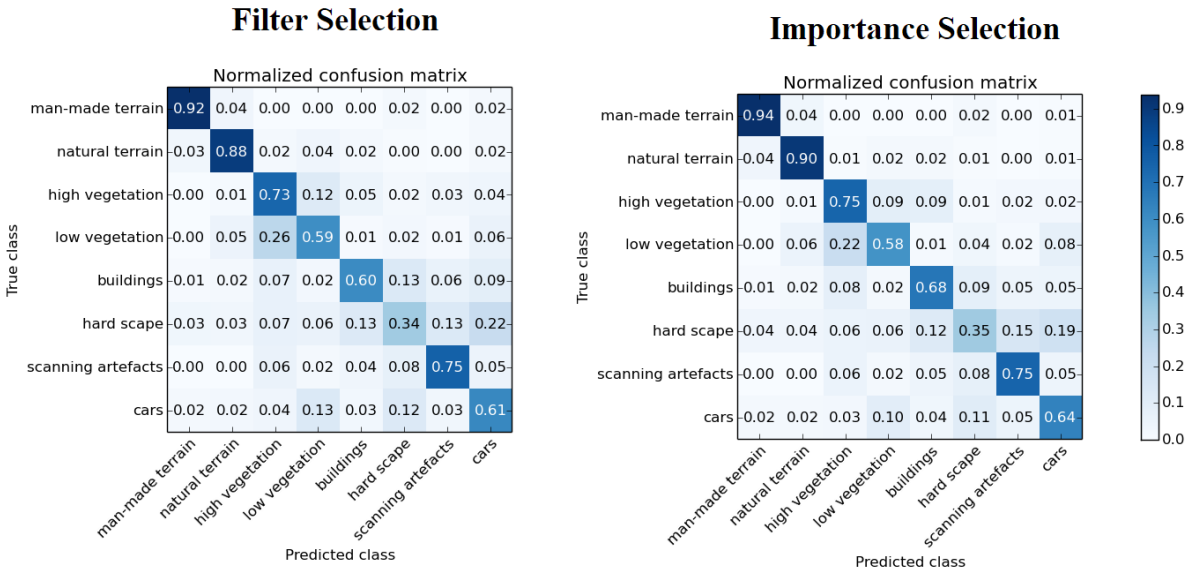
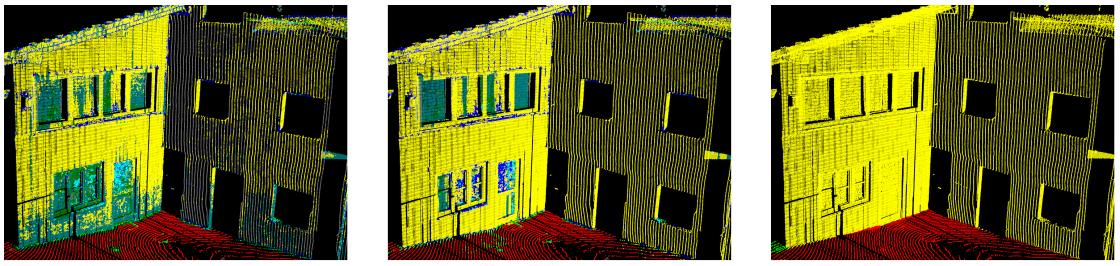


FIGURE 7.10: Normalised confusion matrices of RF classification on Semantic3D dataset, using filter selection features (left) and importance selection features (right).

To determine where this difference lies, Figure 7.10 shows two normalised confusion matrices of a RF trained on each feature selection to classify the Semantic3D test set. The diagonals of these matrices (the accuracy on each class) are similar; for 7 classes, the maximum difference in accuracy is 3%. However, the importance set has 8% higher accuracy than the filter set on the *Buildings* class. This makes sense since the H_{range} feature describes the height difference of points in an upright cylinder and is intended to describe tall vertical objects like walls or poles. The *Buildings* class is one of the largest in the dataset, this 8% therefore explains much of the 4% difference in overall accuracy.



(a) **(b)** **(c)**

FIGURE 7.11: Classification of Semantic3D region after training a RF on different feature sets: **(a)** filter selection features, **(b)** importance selection features, **(c)** ground truth reference.

To visualise the difference in accuracy on the *Buildings* class, Figure 7.11 shows a region of a classified Semantic3D scan for the filtered selection (a), importance selection (b) and the ground truth (c). The filtered selection misclassifies many *Building* (yellow) points as *Scanning artefacts* (grey) or *Hardscape* (teal), whereas the importance selection correctly classifies most *Buildings* points. Without H_{range}, the filtered selection likely confuses *Buildings* with *Hardscape* and *Scanning artefacts* (13% and 6% false positives, per Figure 7.10) as it may lack the vertical context needed to differentiate them. The points belonging to windows are challenging for both selections. This could be explained by how the windows are receded into the wall. Due to this, height features derived from a cylinder with a small radius (5cm) may be less helpful since the wall points are not captured by the cylinder.

7.2.2 Classifiers

Training and prediction speeds

To evaluate classifier speed, Table 7.6 shows the training and prediction speed of the RF, SVM and MLP classifiers on each dataset. Separate timings are given for each feature set: all, filtered and important. Since speed greatly depends on classifier design and parameters — i.e. they do not represent all classifiers of a particular type — implementation details are considered when comparing results.

a) Montelupo

	$\mathbf{All_{train}}$	$\mathbf{Filt_{train}}$	$\mathbf{Imp_{train}}$	$\mathbf{All_{predict}}$	$\mathbf{Filt_{predict}}$	$\mathbf{Imp_{predict}}$
RF	0.99	0.64	0.62	83.69	81.39	79.80
SVM	12.90	4.54	4.38	256.52	165.63	130.06
MLP	0.26	0.21	0.14	8.58	4.72	4.73

b) Songo Mnara

	$\mathbf{All_{train}}$	$\mathbf{Filt_{train}}$	$\mathbf{Imp_{train}}$	$\mathbf{All_{predict}}$	$\mathbf{Filt_{predict}}$	$\mathbf{Imp_{predict}}$
RF	0.96	0.63	0.60	111.02	107.58	106.18
SVM	12.53	4.81	4.98	340.13	123.26	121.87
MLP	1.00	0.19	0.21	11.27	6.35	6.27

c) Oakland

	$\mathbf{All_{train}}$	$\mathbf{Filt_{train}}$	$\mathbf{Imp_{train}}$	$\mathbf{All_{predict}}$	$\mathbf{Filt_{predict}}$	$\mathbf{Imp_{predict}}$
RF	1.05	0.69	0.67	10.73	10.61	10.60
SVM	20.73	7.16	7.16	81.54	31.38	42.73
MLP	0.74	0.44	0.37	1.63	0.92	0.91

d) Semantic3D

	$\mathbf{All_{train}}$	$\mathbf{Filt_{train}}$	$\mathbf{Imp_{train}}$	$\mathbf{All_{predict}}$	$\mathbf{Filt_{predict}}$	$\mathbf{Imp_{predict}}$
RF	3.78	2.35	2.26	159.09	158.56	152.85
SVM	66.54	25.53	23.43	2040.46	824.47	781.01
MLP	1.03	0.73	0.46	17.48	9.81	9.82

TABLE 7.6: Time (in seconds) of each classifier's training and prediction phases for each feature set, grouped by dataset.

The results show that across all four datasets, the MLP classifier had the fastest training and prediction speeds for all feature sets. The SVM was the slowest by a large margin, while the RF had training speeds similar to the MLP but much slower prediction speeds, although not as slow as the SVM. For example, on Semantic3D (Table 7.6.d) with all features the training times of the MLP, RF and SVM were 1, 4 and 67 seconds respectively, while the prediction times were 17, 159 and 2040 seconds. The MLP's fast

train speed is largely due to its shallow design; only one hidden layer which contains $(n\_features + n\_classes)/2$ neurons. In addition, MLPs are inherently fast predictors due to the low cost of passing test samples through the network. SVM training times are increased by a factor of 5 due to 5-fold cross validation, but would still be the longest without it. The SVM's training and prediction is also particularly slow on multiclass datasets (Tables 7.6.c and 7.6.d) due to the implementation's one-against-one approach. The RF training times are close to the MLP since decision trees are simple to train. However, prediction times are around 10 times slower. RF prediction speed is proportional to the number of trees in the forest — if the implementation trained 50 trees instead of 100, prediction would be twice as fast.

The results in Table 7.6 also show the impact of feature set size on training and prediction speeds. The complete feature set (*All*) has 72 features while the important (*Imp*) and filtered (*Filt*) sets have 25 features. The SVM and MLP are much faster when using reduced sets. Using reduced sets, the SVM trains and predicts in 30% - 40% of the time it takes using all features while the MLP takes approximately 50% of time. This is due to a large decrease in complexity when using fewer features; the SVM benefits from mapping samples to a lower dimension hyperspace, while the MLP is accelerated by fewer connections between neurons. The effect of smaller sets on the RF's times is less noticeable. In particular, prediction speeds are at most 5% faster. This is because RF prediction requires the same number of trees (100) to be traversed when classifying a sample, regardless of the number of features.

Classification accuracy

To begin assessing classification accuracy, Tables 7.7 - 7.10 show the recall and precision of the classifiers on each class in the test sets. As a reminder, recall is the proportion of points in a class C that are correctly classified, whereas precision is the proportion of points classified as C that are correctly classified. These are important to differentiate, since recall indicates how well the classifier predicts points in C and precision indicates how often the classifier is correct when it predicts a point as C.

Montelupo

Recall		
	Keep	**Discard**
RF	0.8722	**0.8905**
SVM	**0.8816**	0.8447
MLP	0.8276	0.8448
Precision		
	Keep	**Discard**
RF	**0.8689**	0.8934
SVM	0.8252	**0.8956**
MLP	0.8160	0.8550

TABLE 7.7: Class precision and recall values of each classifier (**Montelupo**).

Songo Mnara

Recall		
	Keep	**Discard**
RF	0.8922	**0.9346**
SVM	**0.8927**	0.9093
MLP	0.8715	0.8897
Precision		
	Keep	**Discard**
RF	**0.9309**	**0.8978**
SVM	0.9067	0.8956
MLP	0.8864	0.8752

TABLE 7.8: Class precision and recall values of each classifier (**Songo Mnara**).

On the binary datasets (tables 7.7 and 7.8), there is little disparity between the precision P and recall R of any classifier. In most cases, a classifier's P and R are within 0.02 of each other. However, on Montelupo (7.7) the SVM has $R - P \approx 0.06$ (*Keep*) and $P - R \approx 0.05$ (*Discard*). This means that the classifiers, except for the SVM, correctly predicted points in C about as often as they were correct when they classified a point as C.

Oakland

Recall					
	Ground	**Building**	**Vegetation**	**Pole**	**Wire**
RF	**0.9615**	0.4974	**0.8059**	0.7439	0.7761
SVM	0.9501	0.4956	0.7549	0.7174	0.7713
MLP	0.9614	**0.5888**	0.7072	**0.7633**	**0.7861**
Precision					
	Ground	**Building**	**Vegetation**	**Pole**	**Wire**
RF	0.9886	**0.7988**	0.8829	**0.1140**	**0.0340**
SVM	**0.9960**	0.7976	0.8762	0.1107	0.0221
MLP	0.9896	0.7983	**0.9176**	0.0840	0.0326

TABLE 7.9: Class precision and recall values of each classifier (**Oakland**).

The multiclass Tables (7.9 and 7.10) show larger differences between R and P. On the *Pole* and *Wire* classes of Oakland (Table 7.9), the classifiers have an average R of 0.74, 0.78 and average P of 0.10, 0.03 respectively. This means that *Pole* and *Wire* points are classified correctly most of the time, but the classifier is rarely correct when it classifies a point as one of these classes. This is common with smaller classes in imbalanced datasets since false positives are more likely and have a larger impact on P. On Semantic3D's

(Table 7.10) *Building* class, the average R and P are 0.63 and 0.91 respectively. Although 37% of *Building* points were incorrectly classified, the classifiers are correct 91% of the times they predict a point as *Building*. This indicates that the classifiers confuse *Building* points with others, as shown by the confusion matrices (Figure 7.10) used to evaluate feature sets, which can occur when there are similar classes in the dataset.

Semantic3D

Recall								
	Man-made terrain	**Natural terrain**	**High vegetation**	**Low vegetation**	**Building**	**Hardscape**	**Artefacts**	**Cars**
RF	0.9455	**0.9008**	**0.7735**	0.5934	**0.6962**	0.3789	0.7553	**0.6438**
SVM	0.9333	0.8688	0.6768	0.6103	0.6608	**0.4075**	**0.7654**	0.5539
MLP	**0.9480**	0.8643	0.6765	**0.6679**	0.5571	0.3690	0.7367	0.4092
Precision								
	Man-made terrain	**Natural terrain**	**High vegetation**	**Low vegetation**	**Building**	**Hardscape**	**Artefacts**	**Cars**
RF	**0.9318**	**0.8939**	0.7389	**0.6100**	**0.9144**	0.2776	**0.1201**	**0.2318**
SVM	0.9106	0.8811	0.7188	0.5446	0.9125	0.2693	0.0830	0.1939
MLP	0.8743	0.7673	**0.7546**	0.5080	0.8989	**0.2833**	0.1059	0.1232

TABLE 7.10: Class precision and recall values of each classifier (**Semantic3D**).

Using recall or precision alone, comparing classifiers is limited. This is because a classifier may have high recall but low precision, or vice-versa. While this can reveal class bias and confusion, metrics like overall accuracy (the proportion of correct classifications) and F1 score (a combination of recall and precision) are more suitable for general comparison.

Table 7.11 summarises the overall accuracy (OA) of each classifier and feature set combination. The results show that the RF paired with all features consistently has the highest overall accuracy on all datasets. Using all features, the RF's average OA is 86.3%, which is 2.1% higher than the average OA of the SVM (84.2%) and 3.8% higher than the MLP (82.5%). The difference is smaller when averaging the OA of all three feature sets, where the RF has 85.5% average OA which is almost 3% higher than both the SVM (83%) and MLP (82.8%). However, the RF's OA with all three feature sets is still higher than the other classifiers paired with any feature set. This gives a general impression of the classifiers' accuracy.

	$\mathbf{RF_{all}}$	$\mathbf{RF_{filter}}$	$\mathbf{RF_{imp}}$	$\mathbf{SVM_{all}}$	$\mathbf{SVM_{filter}}$	$\mathbf{SVM_{imp}}$	$\mathbf{MLP_{all}}$	$\mathbf{MLP_{filter}}$	$\mathbf{MLP_{imp}}$
Montelupo	**0.882**	0.873	0.882	0.861	0.848	0.842	0.837	0.855	0.840
Songo Mnara	**0.914**	0.903	0.909	0.901	0.818	0.890	0.881	0.896	0.890
Oakland	**0.885**	0.881	0.879	0.867	0.879	0.866	0.875	0.873	0.868
Semantic3D	**0.772**	0.723	0.761	0.738	0.708	0.748	0.707	0.700	0.718
Avg	**0.863**	0.845	0.858	0.842	0.813	0.837	0.825	0.831	0.829
Classifier Avg	**0.855**			0.8306			0.8284		

TABLE 7.11: Overall accuracy of each classifier and feature set combination for all four datasets. Highest overall accuracy in bold.

To reveal which classes are the most challenging for each classifier, Figures 7.12.a - 7.12.d show their class F1 scores. Since the F1 score is the harmonic mean of recall and precision, i.e. $\text{F1} = 2 * \frac{P*R}{P+R}$, it follows that classes with low F1 scores have low P, low R or both. This is evident by the low *Pole* and *Wire* F1 scores in Figure 7.12.b, which have a very low average precision (0.10 and 0.03).

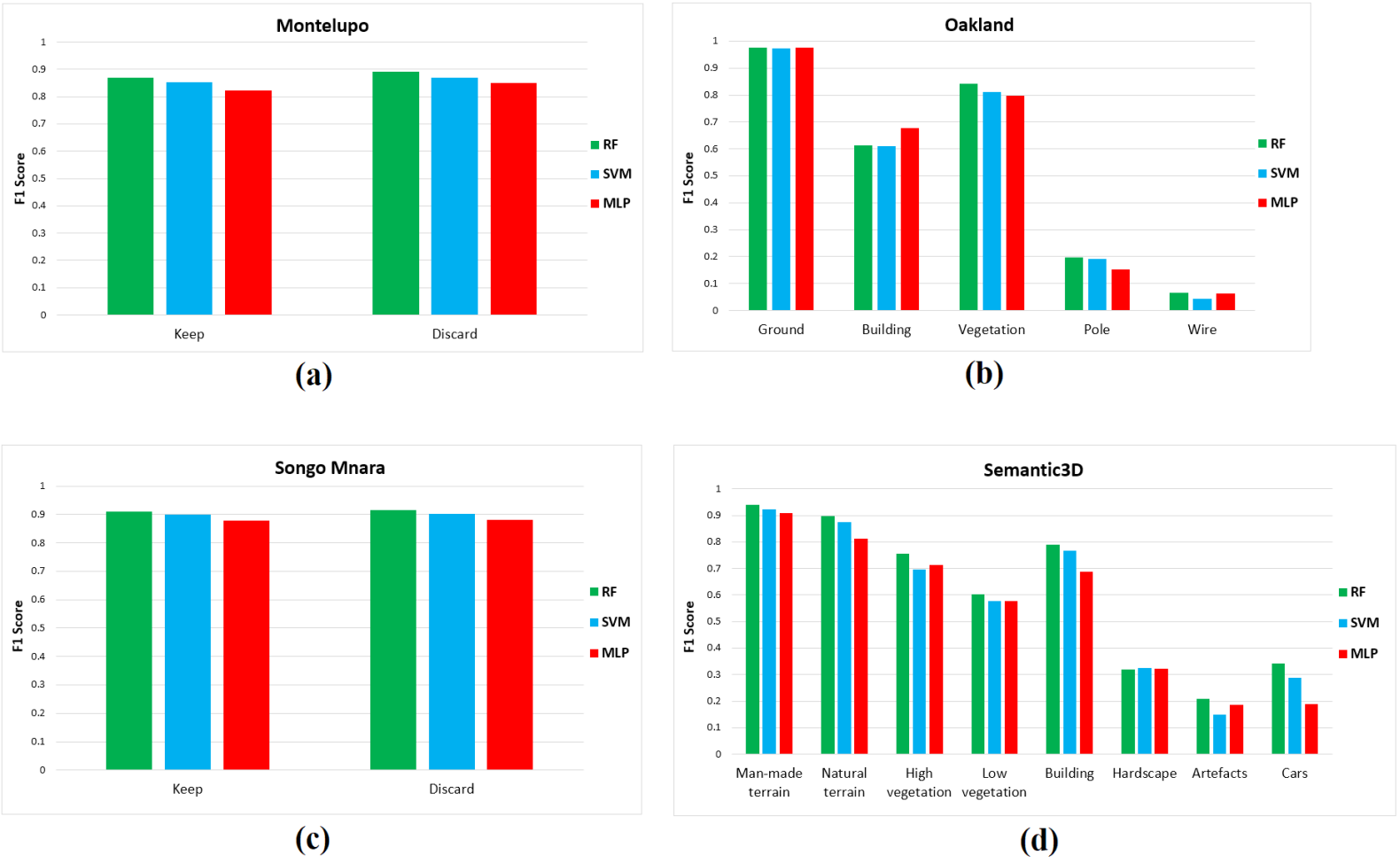


FIGURE 7.12: Classifiers' mean F1 scores for each class and dataset: **(a)** Montelupo, **(b)** Oakland, **(c)** Songo Mnara, **(d)** Semantic3D. Bars are clustered by class and coloured by classifier.

On the binary datasets (Figures 7.12.a and 7.12.c), the RF has the highest F1 score on all classes, followed by the SVM and the MLP. However, on the multiclass datasets (Figures 7.12.b and 7.12.d), the ranking of classifier F1 scores is less consistent. For example, on Oakland (Figure 7.12.b) the MLP scores the highest on the *Building* class but the lowest on almost all others. Similarly, on Semantic3D (Figure 7.12.d) the RF has the highest score on every class except *Hardscape*, where it scores the lowest. To further illustrate the differences between the classifiers' results, two highlighted examples are presented next.

Highlighted example 1: Montelupo

The classification of a Montelupo scan is presented for each classifier in Figure 7.13: (a) MLP, (b) SVM and (c) RF. The ground truth of the scan (d) is also provided. On this dataset, the MLP, SVM and RF achieve 84%, 86% and 88% OA respectively.

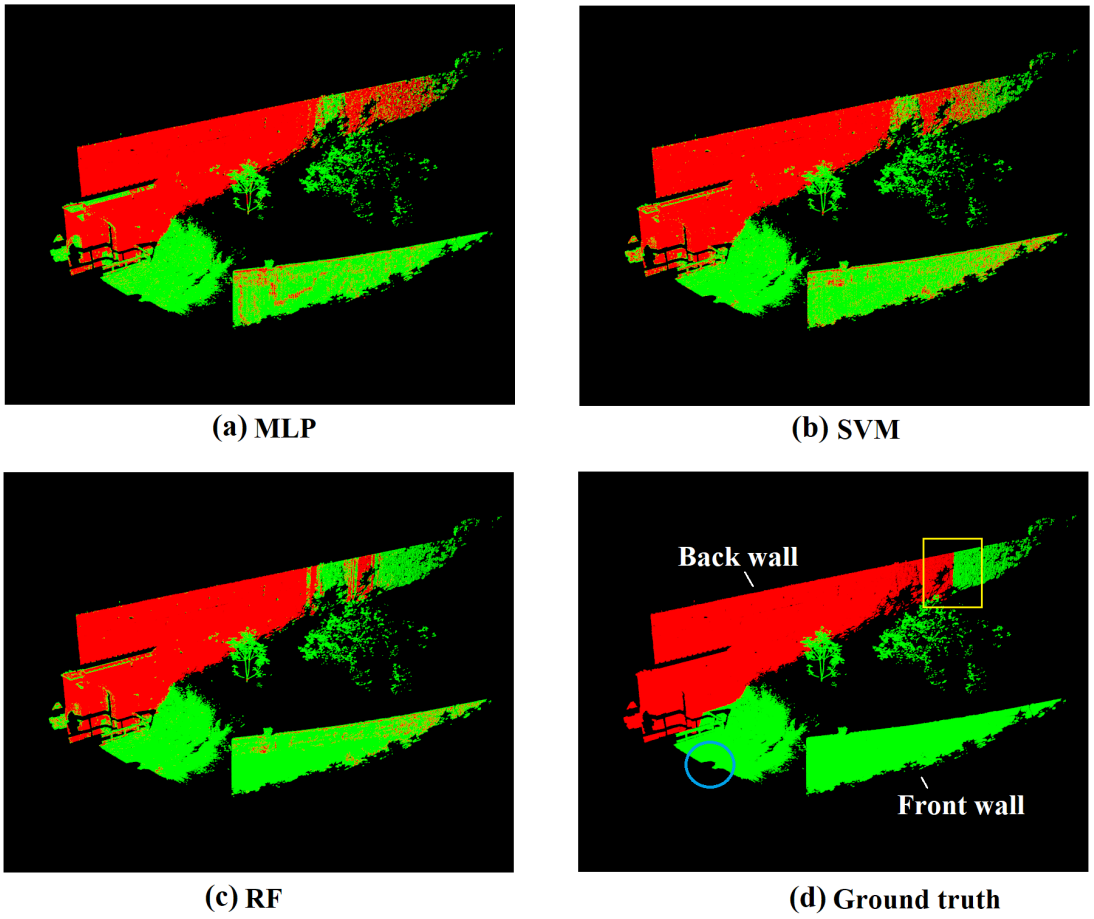


FIGURE 7.13: Classification of a Montelupo scan with each classifier: **(a)** MLP, **(b)** SVM, **(c)** RF, **(d)** ground truth. Keep class = red, discard class = green. Yellow square contains arbitrary class boundary, blue circle shows scanner's position. Classifiers use complete set of features for training and prediction.

As with other scans in the dataset, the ground truth (d) contains arbitrary class boundaries. This is because points are labelled as *Keep* or *Discard* rather than traditional classes like *Wall* or *Vegetation*. Due to this, points belonging to the same object or the same type of object can have different labels. For example, the back wall in the scan is mostly labelled *Keep*, while the front wall is fully labelled as *Discard*. In addition, the same site object or structure may appear in multiple scans but with different labels.

The results show a noticeable difference between the classifiers' predictions. For example, the MLP has the most errors on the front wall and the left side of the back wall. The

SVM classifies the left side more accurately than the MLP, while the RF has the highest accuracy. These points are consistently labelled throughout the dataset, meaning that these results reflect the classifiers' accuracy on normal, expected test points.

Results on other areas are affected by inconsistent labelling, for example: the arbitrary class boundary on the back wall (indicated by the yellow square). The right side of this boundary is accurately classified by the RF, but poorly classified by the other classifiers. The opposite occurs to the left of the boundary, where the RF has the most errors. This may be due to this wall appearing in other scans where the class boundary occurs further to the left, and suggests that the RF is the best at recognising objects/points co-occurring in multiple scans. Although this leads to errors where boundaries are shifted, the RF's improved ability to recognise previously seen areas could explain its higher overall accuracy.

Lastly, a classifier's statistical accuracy is affected its performance at varying point densities. Although scans are downsampled to one point per 2.5cm^3, points furthest away from the scanner are much sparser. It follows that results from areas closer to the scanner have a larger impact on statistical accuracy. In the example, many of the RF's incorrect predictions occur around the class boundary (yellow square) on the back wall, which is relatively far from the scanner's position (blue circle). Due to point sparsity, the RF's errors on the back wall have a small impact on overall accuracy, while the MLP and SVM's errors on the dense front wall have a large, negative effect on accuracy. The RF's high accuracy on dense areas and lower accuracy on sparse, low-impact areas further explains its rank above the other classifiers.

Highlighted example 2: Oakland

The normalised confusion matrices of the three classifiers' predictions on the Oakland dataset are shown in Figure 7.14. Although the classifiers achieve similar OA on this dataset (RF 89%, SVM 87%, MLP 88%), their F1 scores on each class are varied. These matrices help explain the classifiers' varying class F1 scores.

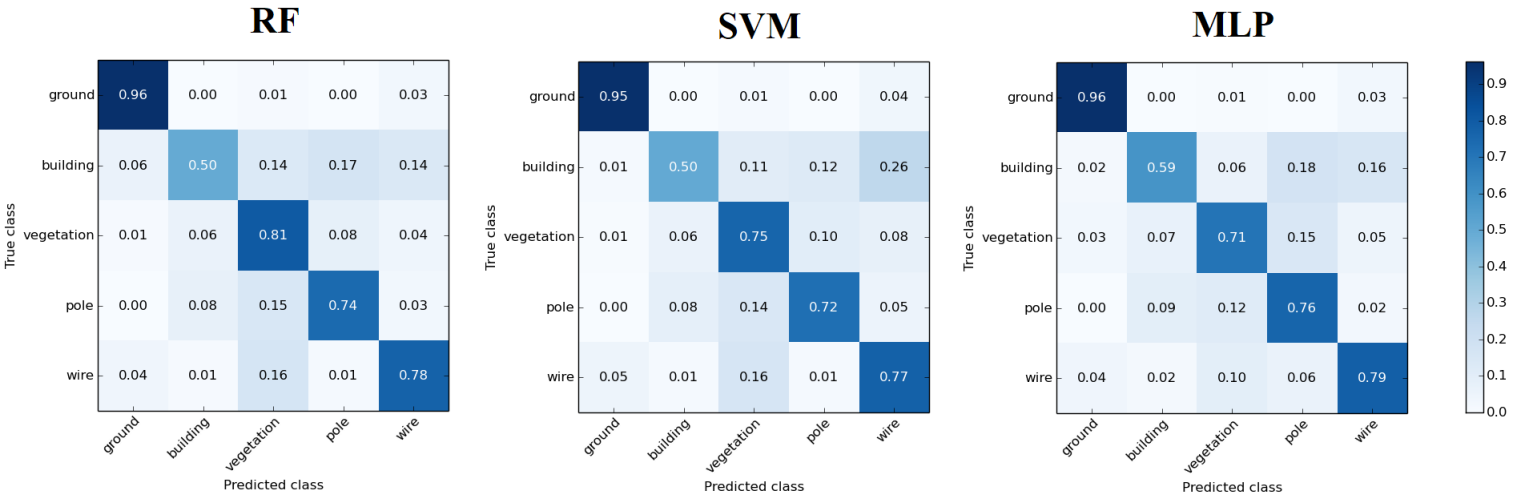


FIGURE 7.14: Normalised confusion matrices of RF, SVM and MLP classifiers' results on Oakland dataset (using all features).

The MLP achieves the highest F1 score (0.68) on the *Building* class, roughly 0.07 higher than the other classifiers. All three classifiers confuse the two smaller classes (*Pole* and *Wire*) with *Building* points fairly often (between 12% - 26% of points). However, on the two larger classes, *Ground* and *Vegetation*, the MLP confuses only 2% of *Ground* points and 6% of *Vegetation* points with *Building* points. Considering class size, this is far fewer points than the RF (6% *Ground*, 14% *Vegetation*) and SVM (2% *Ground*, 11% *Vegetation*) misclassify as *Building*. However, the MLP is also the most likely to incorrectly classify points in other classes as *Building*. This suggests that the MLP is biased towards the *Building* class, improving its accuracy on the class but lowering its overall accuracy on the dataset.

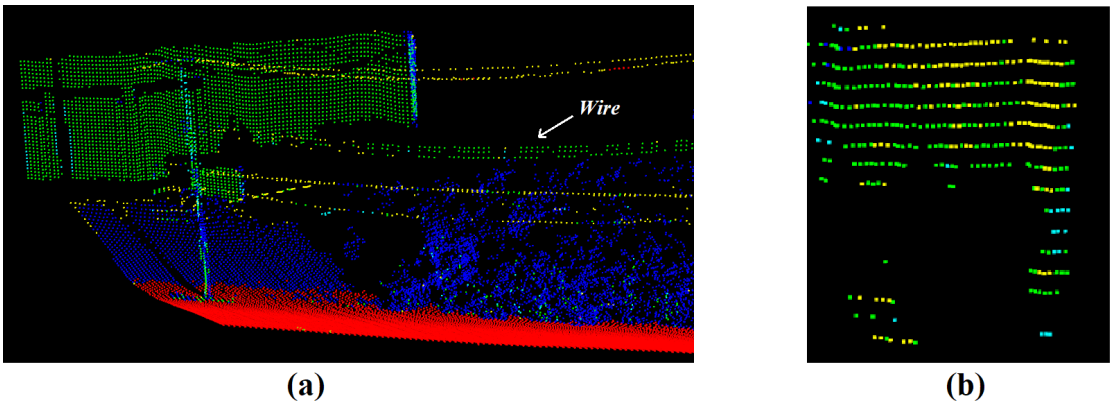


FIGURE 7.15: SVM's confusion between *Building* and *Wire* class. **(a)** *Wire* points classified as *Building*, **(b)** Distant, low density region with *Building* points classified as *Wire*. *Building* = green, *Wire* = yellow.

The SVM has a very low F1 score on the *Wire* class despite usually having the second highest F1 score on other classes. The classifiers have very low precision ($< 4\%$) on this class, meaning that very few of the points predicted to be *Wire* actually are. In particular, the SVM incorrectly classifies *Building* points as *Wire* 26% of the time. The confusion between *Building* and *Wire* is likely due to their similarity at low point densities where walls appear as stacked, horizontal lines. This effect, and the SVM's confusion between the two classes is illustrated in Figures 7.15.a (*Wire* confused as *Building*) and 7.15.b (*Building* confused as *Wire*). This example of large classes being misclassified as smaller classes like *Wire* is indicative of the SVM's lowest overall accuracy on the dataset.

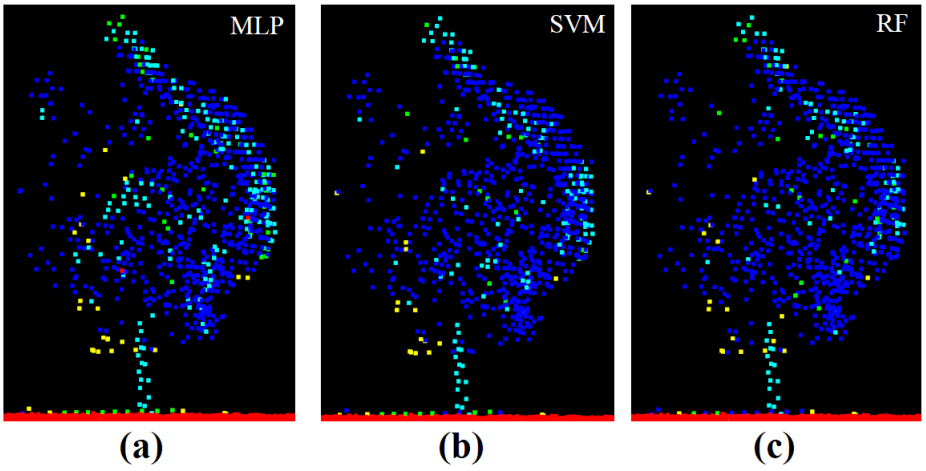


FIGURE 7.16: Classifiers' results on points belonging to an isolated tree in the Oakland dataset: **(a)** MLP, **(b)** SVM and **(c)** RF. *Vegetation* = blue, *Pole* = teal .

Lastly, the RF achieves the highest F1 score on the *Vegetation* class. It also has the highest accuracy on this class at 81%, which is 6% higher than the SVM and 10% higher

than the MLP. *Vegetation* is a large class, and is only sometimes (8%) confused with the *Pole* class by the RF. The MLP confuses *Vegetation* with *Poles* almost twice as often (15%), while the SVM only confuses the two 10% of the time, but misclassifies 8% of *Vegetation* points as *Wire*, twice as often as the RF. Figures 7.16.a - 7.16.c illustrate the classifiers' confusion between the *Vegetation* (blue) and *Pole* (teal) class, where the RF is the most accurate. Overall, the RF's high F1 score on the *Vegetation* class and low confusion rates explains its position as the most accurate classifier on the Oakland dataset.

7.2.3 Scan Selection

The previous results show that the RF and complete feature set were consistently the most accurate combination for classification. The scan selection experiments therefore used this pairing to test random selection and the designed selection schemes. The remainder of this section analyses the speed and accuracy of the schemes, followed by two highlighted examples.

Selection pipeline speed

To evaluate scan selection speed, Table 7.12 shows the processing times of the pipeline's four stages for each dataset. *Segmentation* is the most expensive stage due to the high complexity of the VCCS algorithm, accounting for up to 80% of processing times. The fastest stage, *Scoring*, consistently takes less than 0.1s since calculating balance and similarity scores from cluster assignments is computationally simple. The *Features* stage accounts for roughly 15% of the pipeline, taking up to 6s to extract supervoxel-based features, making it much faster than point-based features which require up to 190s for prediction alone (as shown in Section 7.2.1). The *Clustering* times vary the most, taking between 5% and 27% of the total time, which is likely due to the time needed for k-means to converge.

	Segmentation	**Features**	**Clustering**	**Scoring**	**Total**	**# Scans**	**Avg**
Montelupo	21.17s	4.13s	1.36s	0.01s	26.71s	16	1.67s
Songo Mnara	16.56s	3.32s	7.25s	0.04s	27.17s	15	1.81s
Oakland	1.83s	0.36s	0.57s	0.01s	2.77s	20	0.14s
Semantic3D	19.92s	5.77s	9.61s	0.07s	35.37s	20	1.77s

TABLE 7.12: Time (in seconds) required by the four main scan selection stages, with average selection time per scan.

In total, selection takes around 30s for three of the four datasets tested, while the low-density Oakland dataset takes only 3s. Averaging by the number of scans, it takes less than 2s per scan regardless of dataset. Although no formal testing of manual scan selection was done, first-hand experience with manual selection and discussions with scan operators indicates that manually selecting scans is much slower. Indeed, manually iterating through a dataset to inspect and identify useful scans while simultaneously comparing them to previously seen scans is a lengthy task. For larger scanning campaigns, this is increasingly onerous as there are more scans to recall and comparing them is (conceptually) a n^2 problem. On the tested datasets, a conservative estimate of 20s per manually selected scan means that the proposed method is at least one order of magnitude faster than manual selection. It is even faster on low-density datasets which could be more difficult to manually select from due to their lack of detail.

An alternative to both methods, random selection, was also tested. However, since selecting scans via random indices is essentially instant, no timing measurements are presented. Instead, later sections will determine if the effect of automated selection on classification accuracy is worth the time overhead it introduces.

Segmentation and clustering

To provide insight on how the optimal number of supervoxel clusters is chosen, Figures 7.17.a - 7.17.d show the elbow method graphs for each dataset. As a reminder, the elbow is the point where the reduction of sum of squared errors (SSE) starts to diminish as k (number of clusters) increases.

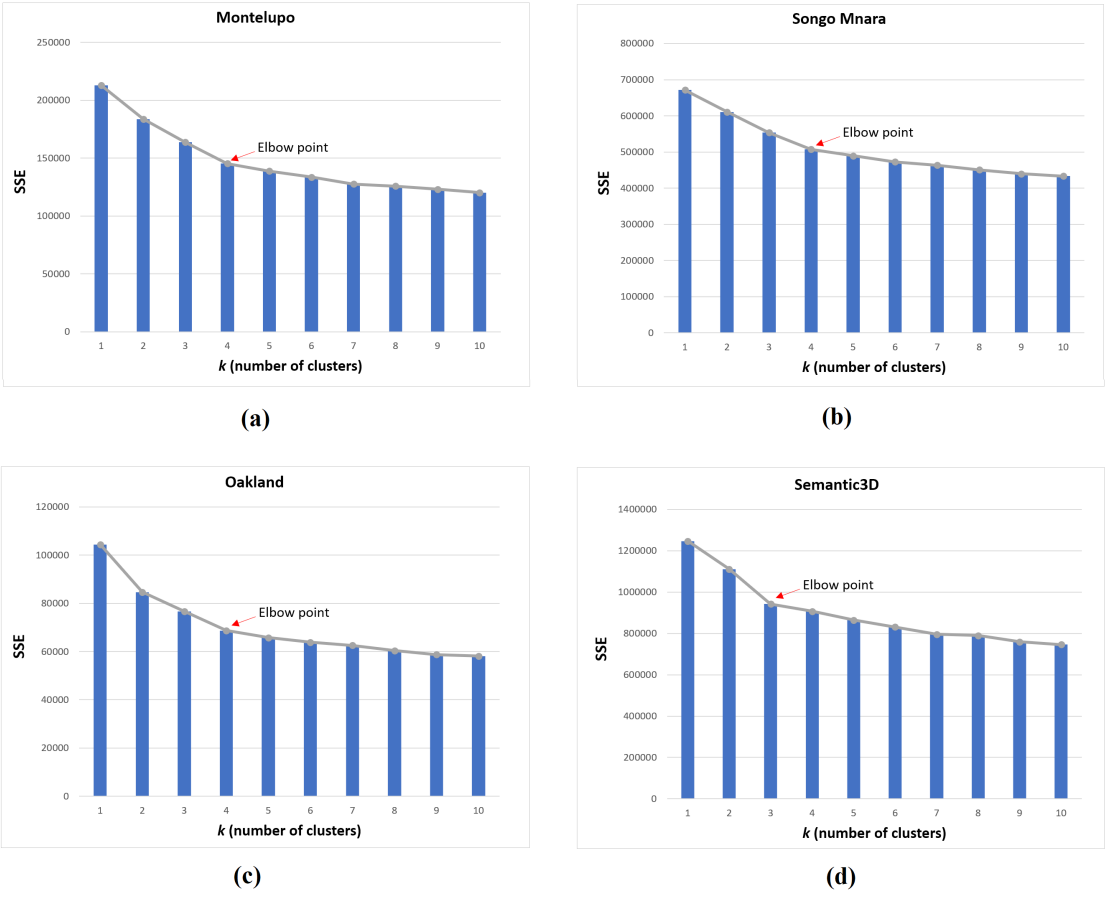


FIGURE 7.17: Elbow method graphs used to find optimal k for k-means supervoxel clustering for each dataset: **(a)** Montelupo, **(b)** Songo Mnara, **(c)** Oakland, **(d)** Semantic3D.

This point is manually identified, meaning the method is prone to human error and not entirely automatic. However, the task is straightforward and quick - typically requiring less than 5 seconds to complete. A more important factor is the segmentation parameters, specifically the voxel resolution R_voxel and seed resolution R_seed of the VCCS algorithm. Depending on how these are set, the supervoxels and the resulting elbow method graphs can change greatly, leading to a different optimal k being chosen. In

practice, choosing k requires iteratively tweaking segmentation parameters until scans are reasonably segmented and a clear elbow point can be identified.

The elbow graphs show that three of the datasets (Figures 7.17.a - 7.17.c) have optimal k=4 and the Semantic3D dataset (Figure 6.7.d) has k=3. This difference could be due to the large variation of scenes in the Semantic3D dataset causing k-means to group supervoxels into three general clusters rather than four more distinct clusters. It could also be due to the particularly large number of Semantic3D segments, resulting in smaller segment "classes" being unable to form their own clusters and instead grouping with more common segments.

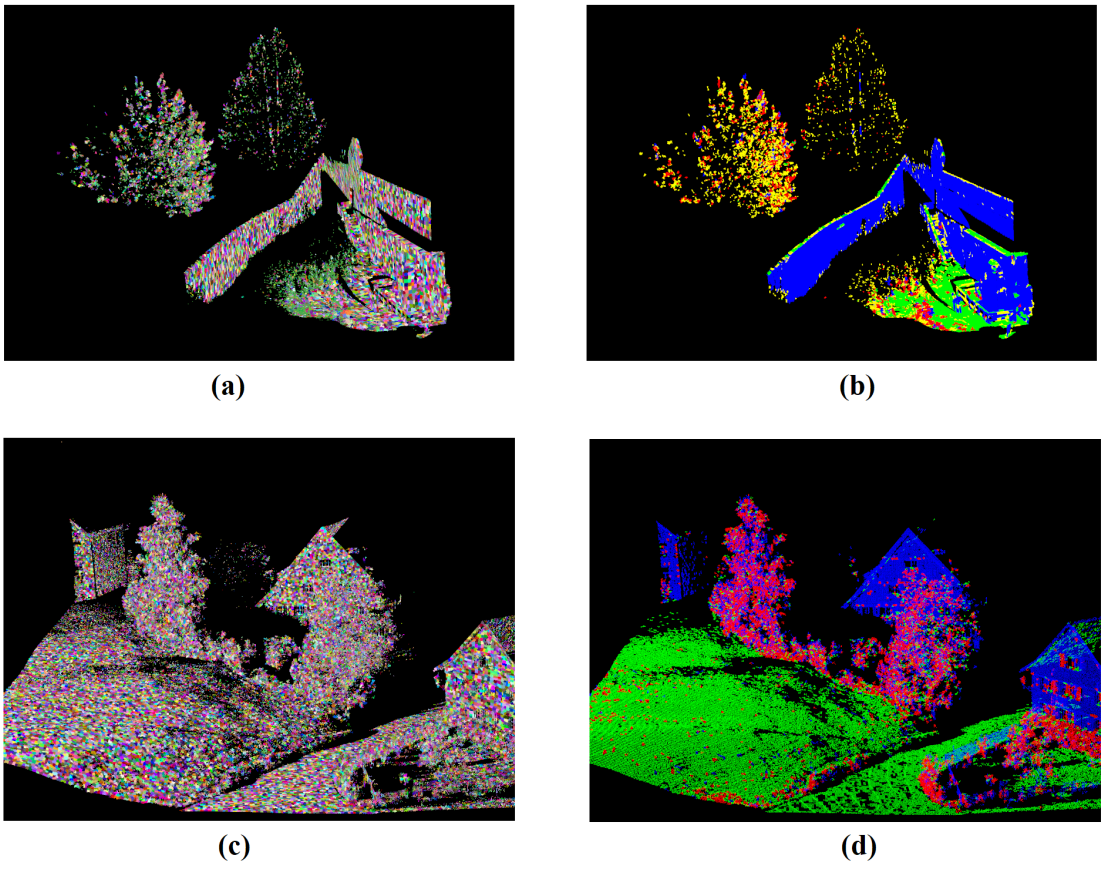


FIGURE 7.18: Supervoxel segmentation and k-means cluster assignments of scans from two datasets: Montelupo segments **(a)** and cluster assignments **(b)**, Semantic3D segments **(c)** and cluster assignments **(d)**.

Visual results from the first two scan selection stages, segmentation and clustering, are presented in Figure 7.18. Note that these are results from two separately processed datasets, and that segments from all scans in a dataset are clustered at once. Segmentation of the Montelupo (Figure 7.18.a) and Semantic3D (Figure 7.18.c) scans is generally good, with most points from clearly different objects (e.g. walls and ground) belonging to different segments. However, points in very sparse areas, e.g. the green points near

the center of Figure 7.18.a, are not close enough to other points to form segments. Given the low number of these points and the limited detail they provide, they are ignored by the clustering step.

The cluster assignment scans (Figures 7.18.b and 7.18.d) show mixed results. Overall, a large proportion of the segments in each cluster belong to the same type of object. For example, in Figure 7.18.d most ground, building and vegetation points are assigned to the green, blue and red clusters respectively. However, in Figure 7.18.b certain segments are clearly in the wrong cluster, such as the horizontal segments on the top of the walls being mistaken for ground (green) segments. Lastly, very small segments (less than 5 points) such as those of the roof in Figure 7.18.d are not clustered since some features require at least 5 points to be extracted.

Despite some limitations, the results of the segmentation and clustering stages are satisfactory and provide enough information for the rest of the pipeline to describe scans and make an informed selection. However, a worst case segmentation or clustering could limit the rest of the pipeline's ability to sensibly compare scans. The effect of clustering, and other factors, on a selection's classification accuracy is therefore analysed in the following sections.

Effect of schemes on classification accuracy

Before comparing the RF's accuracy when trained on selected or random scans, it is important to analyse the random runs alone. Table 7.13 shows the RF's overall accuracy (OA) when training on random scans, with seven runs per dataset. The standard deviation of the OA is also given.

	$\mathbf{Rand_1}$	$\mathbf{Rand_2}$	$\mathbf{Rand_3}$	$\mathbf{Rand_4}$	$\mathbf{Rand_5}$	$\mathbf{Rand_6}$	$\mathbf{Rand_7}$	σ
Montelupo	0.8774	0.8849	0.8659	0.7324	0.8517	0.8381	0.8874	0.0541
Songo Mnara	0.9206	0.8913	0.9088	0.9118	0.9062	0.9186	0.9018	0.0101
Oakland	0.9113	0.9076	0.9196	0.9098	0.9090	0.9088	0.9046	0.0047
Semantic3D	0.7823	0.7416	0.7439	0.7363	0.6110	0.7329	0.6982	0.0543

TABLE 7.13: Overall accuracy and standard deviation of RF classifier trained on random scan selections, 7 runs per dataset.

The datasets with the largest deviation in OA (5.4%) are Montelupo and Semantic3D. This is due to number of factors. Firstly, the Montelupo dataset has six scans with many more points than other scans due to the resolution they were originally captured at. Secondly, the Semantic3D dataset is the most varied of the four tested, with scans spanning large areas from different sites. Lastly, both datasets contain at least one very small, uninformative scan; a result of splitting the original scans by their X-coordinate signs. While undesirable, these "anomaly" scans are included to test if the selection

schemes avoid them. Given these flaws and the lack of homogeneity between scans, these datasets are therefore challenging and the deviation in OA yielded by random training selections is understandable.

On the Songo Mnara and Oakland datasets however, there is a much smaller deviation (no more than 1%) in OA between random runs. Scans in these datasets are more homogeneous, with similar objects and composition, as well as less significant size differences. Furthermore, neither of these datasets contain any particularly problematic scans following X-coordinate splitting. Lastly, the RF consistently achieves a higher OA (around 90%) on these datasets than on the others. This suggests they may be inherently simpler to model, reducing the need for highly descriptive scans and reducing the impact of changes to the training set. The small deviation in accuracy between random runs on these two datasets is therefore explained by their homogeneity and relative simplicity.

To compare the accuracy of the selection schemes to random selection, Figures 7.19 and 7.20 below show the OA achieved by the three schemes (*balanced*, *similarity*, *distinct*) on each dataset as well as the random runs' minimum, maximum and average OA. The comparison is split into two parts. First, selection schemes are compared to the random runs on the challenging, less homogeneous datasets that saw the most variation in random run accuracy (Montelupo and Semantic3D). Then, comparisons are made on the simpler, homogeneous datasets that the random selections consistently achieved high OA on (Songo Mnara and Oakland).

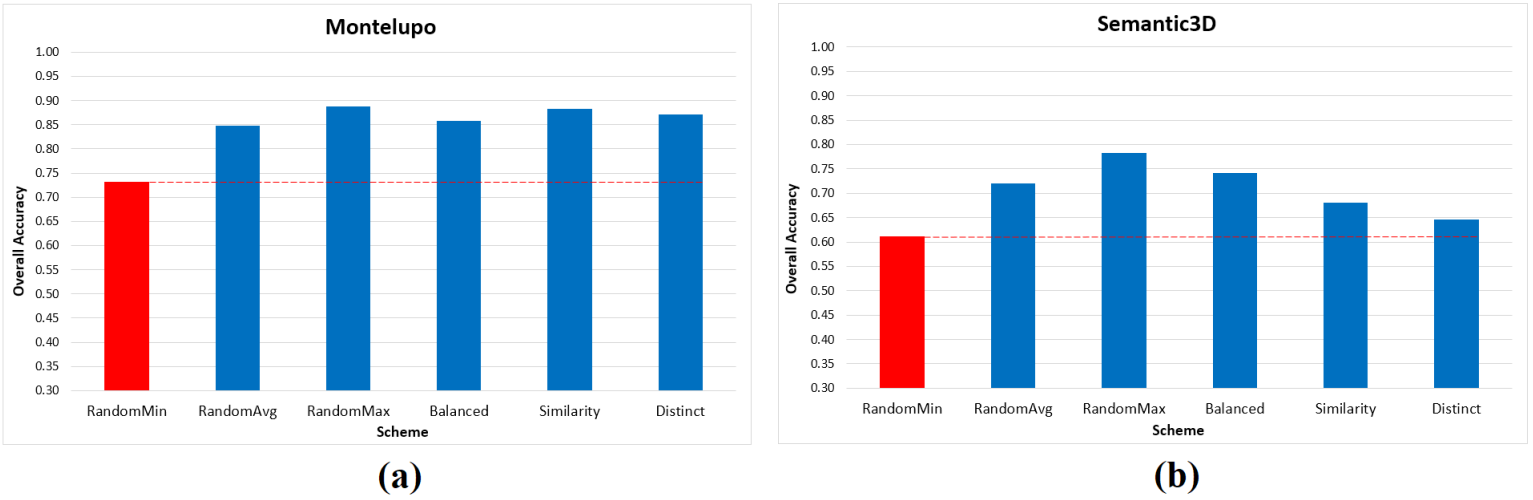


FIGURE 7.19: Scan selection schemes' and random runs' overall accuracy on **(a)** Montelupo and **(b)** Semantic3D. The least accurate random run (RandomMin) is highlighted red.

The success of the selection schemes is most evident on the challenging datasets. For example, on the Montelupo dataset (Figure 7.19.a) all three schemes achieve a higher (up to 3%) OA than the average random run, and a much higher (13 - 15%) OA than

the worst random run. In particular, the *similarity* scheme is within 1% of the highest random run's OA. The Semantic3D dataset (Figure 7.19.b) shows a large variation in OA between the schemes and the random runs. Although only the *balanced* scheme is more (2%) accurate than the average random run, all three schemes achieved a much higher (4 - 13%) OA than the lowest random run. Since this is the lowest of only 7 runs, it is likely that other untested random combinations could be even less accurate. In such a case, the improvement of the schemes over the random minimum would be even greater since the schemes' selections would remain unchanged. This highlights a key advantage of the schemes — deterministic selection. Rather than leaving the accuracy of a classifier to chance by randomly selecting a set of potentially weak training scans, the schemes have shown to consistently outperform the worst random runs on challenging campaigns and provide more confidence in the quality of the training set.

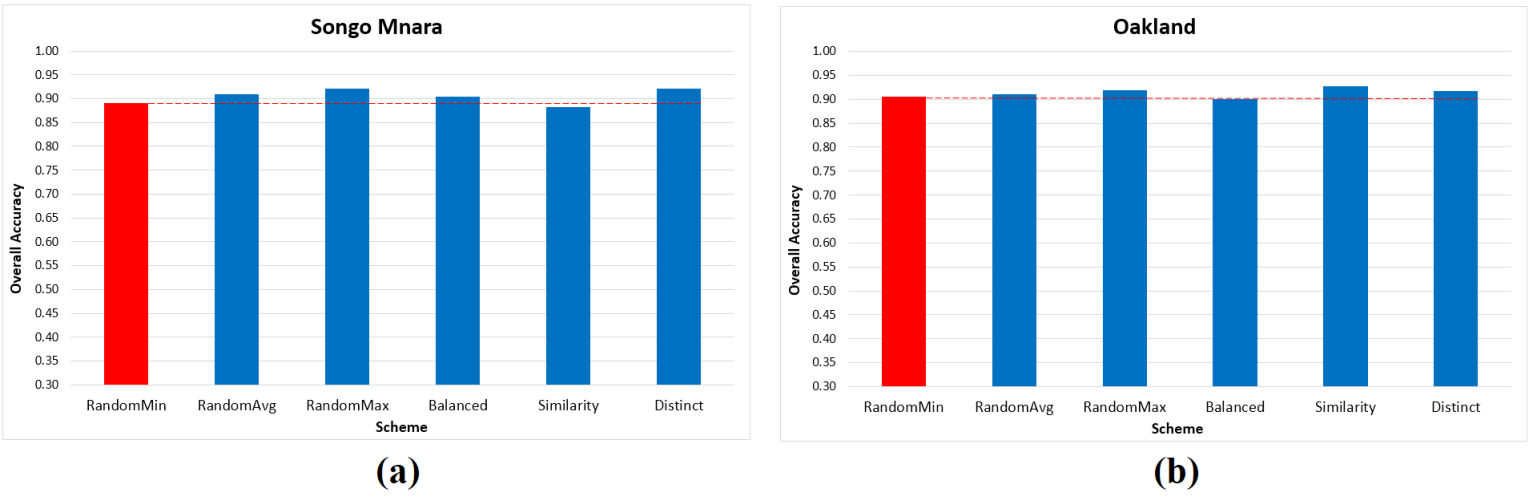


FIGURE 7.20: Scan selection schemes' and random runs' overall accuracy on **(a)** Songo Mnara and **(b)** Oakland. The least accurate random run (RandomMin) is highlighted red.

The results on the less challenging datasets show a smaller impact of the selection schemes. For example, on Songo Mnara (Figure 7.20.a) and Oakland (Figure 7.20.b), some schemes score higher than the average run while others score lower, although the difference is minor (less than 2%). At least two of the schemes score higher than the worst random run on both datasets, although also by a very small amount. Overall, there is little comparison to be made since the selections (random or scheme) score consistently high and within approximately 3% of each other. This is expected since these are the most homogeneous and least challenging datasets which, coupled with a large number of training scans, greatly reduces the importance and impact of scans used for training. These results provide contrast to the previous findings, revealing that the schemes are most beneficial when datasets are challenging.

The OA graphs also reveal how the selection schemes compare to each other. Most notably, there is no scheme that consistently scores the highest or lowest of the three. Instead, each scheme's performance relative to the others varies by dataset. For example, the *similarity* scheme has the highest OA on Montelupo and Oakland, but the lowest on Songo Mnara. The *distinct* scheme scores the highest on Songo Mnara, but scores particularly (up to 9 %) lower than the other schemes on Semantic3D. There is a consistent ranking for Montelupo and Oakland, where *similarity* > *distinct* > *balanced*, but this does not extend to the other datasets.

Discussion

There are a number of potential reasons for the selection schemes' varying levels of improvement over the random runs and each other. Firstly, if a dataset's scans are homogeneous with little variation between random runs, the selection schemes' impact may be minimised since changes to the training set have an insignificant effect on the classifier. Conversely, if a dataset is inhomogeneous then changes to the training set may have a high impact on classification accuracy.

Secondly, the large training set size (20 - 25% of the dataset) may greatly benefit random selections as they have a higher chance of including useful scans. If the training sets were smaller (e.g. 10% of the dataset), the probability of the best scans being randomly selected would be lower. However, the selection schemes would still select the highest scoring scans. In such a scenario, the difference between the random runs and the selection schemes may be more noticeable.

Lastly, the selection schemes depend on the descriptiveness of balance and similarity scores, which may be unreliable due to poor segmentation or clustering. Even when scans are described well by these scores, a scheme's core principle may have a negative effect. For example, a small, noisy scan with limited information might be distinct (and thus chosen by the distinct scheme) but it is not suitable for training. To explain these reasons further, the next section presents example selections and discusses why they were chosen as well as their potential impact on classification accuracy.

Highlighted example 1: Montelupo

The Montelupo dataset sees the greatest increase in OA when using the selection schemes instead of random selection (up to 3% higher than the average run and 13-15% higher than the minimum). To explain the increase, this section begins with a discussion of the scans' balance and similarity scores, as well as the distinct scheme's cluster sizes. Then, exemplar scans from each selection scheme are analysed based on their cluster assignments and ground truth.

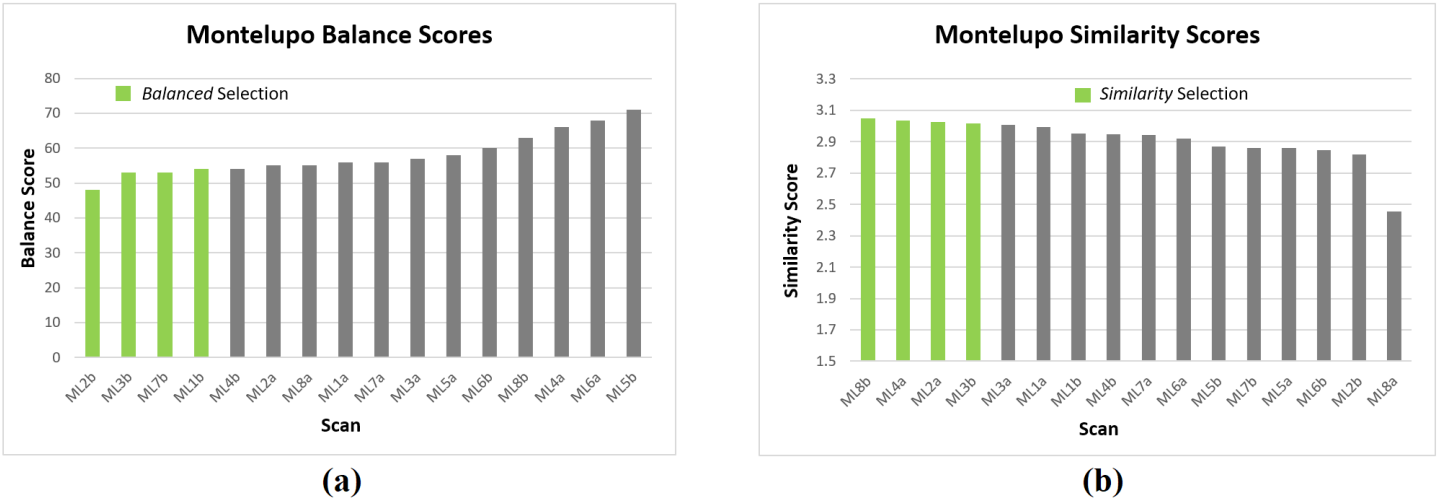


FIGURE 7.21: *Balance* **(a)** and *similarity* **(b)** scores of each scan in the Montelupo dataset. Green bars denote scans selected by the respective scheme.

The balance scores in Figure 7.21.a show that although the scans are imbalanced (much larger than the perfect balance score of 0), their scores are fairly consistent with most falling between 50 and 60. This indicates that the scans in the dataset have an uneven distribution of supervoxels i.e. the types of objects in each scan are not equally represented.

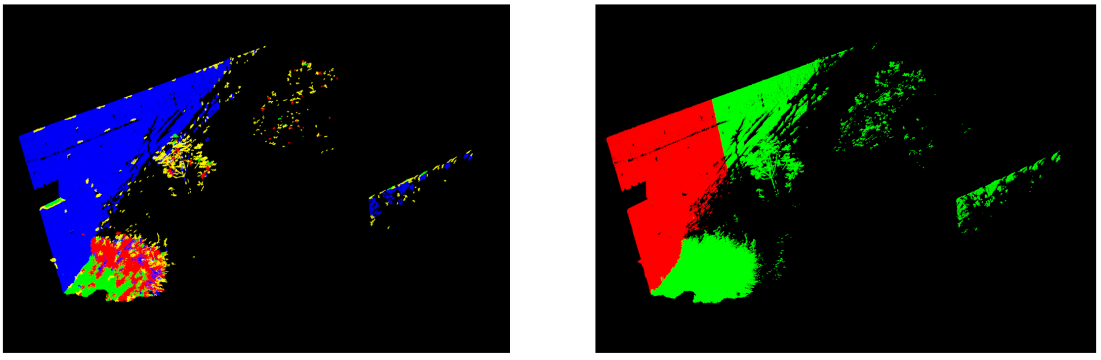
The similarity scores in Figure 7.21.b are not as straightforward to read. Since a scan's similarity score is the sum of its supervoxels' average cosine similarity $\in [-1, +1]$ to the center of their cluster, the perfect similarity score is k, the number of clusters. All of the similarity scores, except one, fall between 2.8 and 3.05, i.e. 70 - 76% of the perfect score (4.0). This means that the objects in each scan (estimated with supervoxels) are reasonably similar to objects of the same type (their clusters) in other scans.

	Montelupo		Songo Mnara		Oakland		Semantic3D	
	Size	**Center**	**Size**	**Center**	**Size**	**Center**	**Size**	**Center**
Cluster 1	8	ML3b	3	SM10a	4	OAK11	1	UNT3a
Cluster 2	6	ML5a	1	SM8a	5	OAK6	9	UNT1b
Cluster 3	1	ML8a	2	SM9a	1	OAK15	1	BIL1d
Cluster 4	1	ML8b	10	SM10b	5	OAK9	1	BIL5b
Cluster 5	—	—	4	SM4a	—	—	8	BIL5a

TABLE 7.14: *Distinct* scheme's cluster sizes and center scans selected for training, per dataset. Smaller datasets are clustered into 4 groups only (cluster 5 not applicable).

Lastly, the distinct scheme's cluster sizes and centers are given in Table 7.14. These are not the clusters that balance and similarity scores are derived from; they are the k-means (k = number of scans to select) clusters found using feature vectors of scans' balance and similarity scores. The table shows two large clusters (size 8 and 6) and two single-scan clusters are found when selecting k=4 Montelupo scans. This means that two of the scans in the dataset are highly distinct, or that two sensible clusters exist and single-scan clusters are a result of forcing k-means to find four non-empty clusters. In either case, the selected scans differ from each other as much as possible while representing their clusters.

Next, an exemplar scan from each selection scheme is analysed based on their cluster assignments and ground truth. The cluster assignments reveal why scans are selected, while the ground truth illustrates how balanced, similar or distinct the scans actually are.

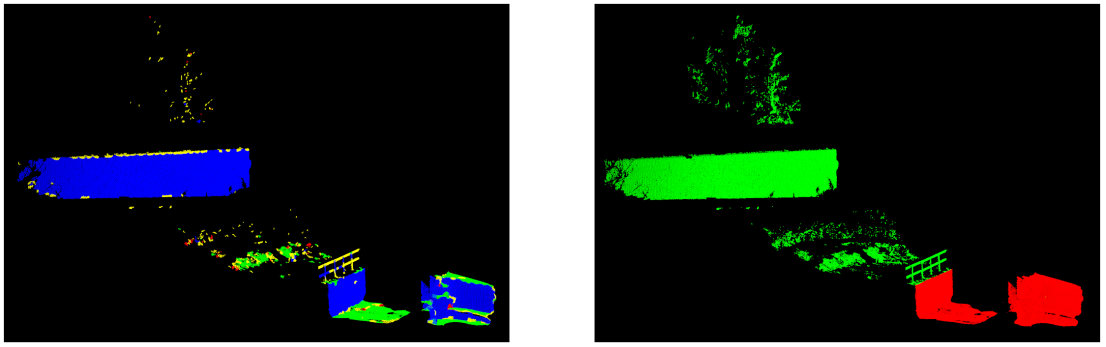


(a) Cluster assignments **(b) Ground truth**

FIGURE 7.22: Exemplar *balanced* scan **ML2b** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

The exemplar balanced scan, ML2b, has the best (lowest) balance score of the Montelupo scans. It therefore contains the most even distribution of supervoxels from each cluster. Although it is not particularly balanced (balance score = 48), the scan contains

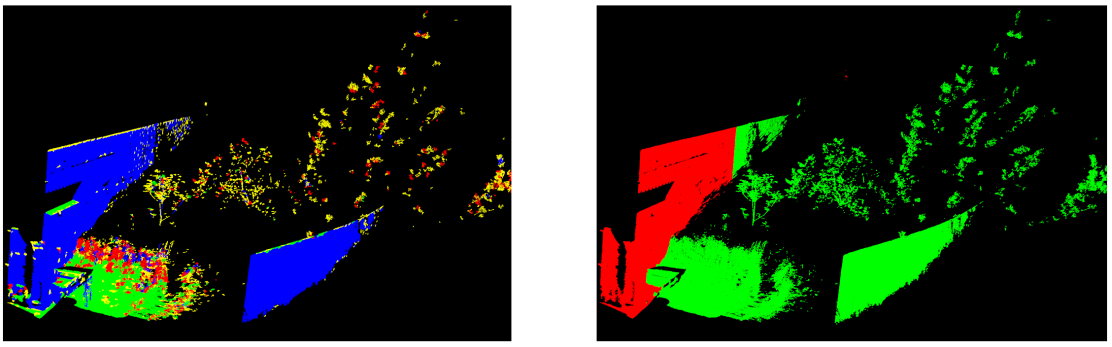
a noticeable spread of supervoxels from each cluster (Figure 7.22.a). Besides the largest cluster (blue) that most wall segments belong to, the remaining clusters are evenly spread between ground (green), low vegetation (red) and high vegetation (yellow) supervoxels. Since walls are the focus of the scanning campaign, it is expected that they form the largest cluster. The ground truth scan (Figure 7.22.b) is similarly balanced in terms of its object composition.



(a) Cluster assignments **(b) Ground truth**

FIGURE 7.23: Exemplar *similarity* scan **ML8b** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

The exemplar similarity scan, ML8b, has the highest similarity score. Based on its cluster assignments (Figure 7.23.a), very few supervoxels belong to the red and yellow cluster, while over 90% belong to the blue and green cluster. In this dataset, red and yellow clusters typically consist of vegetation segments which are more varied than the homogeneous blue (walls) and green (ground) segments. Since clusters are equally weighted when scoring similarity, scan ML8b's high score is not due to a high proportion of blue and green segments, but due to a high sum of individual similarity scores for each cluster. Although it is difficult to gauge how similar the scan's segments are to their clusters by inspection, the ground truth (Figure 7.23.b) is visually similar to other scans in the dataset since most scans contain walls and different types of vegetation.



(a) Cluster assignments **(b) Ground truth**

FIGURE 7.24: Exemplar *distinct* scan **ML3b** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

Scan ML3b is selected by the distinct scheme as it is the closest to the center of one of the k=4 scan clusters. The distinct scheme selects scans that are distinct from *each other* while also representing other scans, not scans that are simply the most distinct in the dataset. Scan ML3b is the center of the largest cluster (size 8), meaning that it represents half the dataset but is distinct from the other three clusters. Its representational strength is evident by how it is the only scan in the dataset selected by all three schemes. Based on its cluster assignments (Figure 7.24.a) and ground truth (Figure 7.24.b), the scan represents a diverse, commonly surveyed area of the campaign but is distinct from other scans in the selection, e.g. ML8b, with different structure and object distribution.

Overall, the process behind the schemes' selections is clearer after inspecting example Montelupo scans' scores and cluster assignments. Although not all the selections are reviewed, the presented ground truth scans indicate that the schemes successfully select balanced, similar or distinct scans. This ability to make intentional, descriptive training set selections instead of relying on chance explains the schemes' increase in classification accuracy over random selections. As this varies between datasets, the next section highlights example scans from the Semantic3D dataset.

Highlighted example 2: Semantic3D

Using the schemes to select training scans from the Semantic3D dataset yields mixed results. While the balanced scheme improves on the lowest random run by 13%, the similarity and distinct schemes improve by 7% and 4% respectively. As before, this section discusses dataset scores and analyses individual selected scans to explain the schemes' performance.

Before discussing scores, it is important to consider the difficulty of clustering Semantic3D supervoxels. Of the four datasets tested, Semantic3D has the most points, classes (8) and contains scans from different sites. Given this complexity, k-means is unlikely to separate supervoxels into uniform clusters. Instead, three general clusters with high intracluster variance are found. This does not invalidate balance and similarity scores but impacts how they are interpreted.

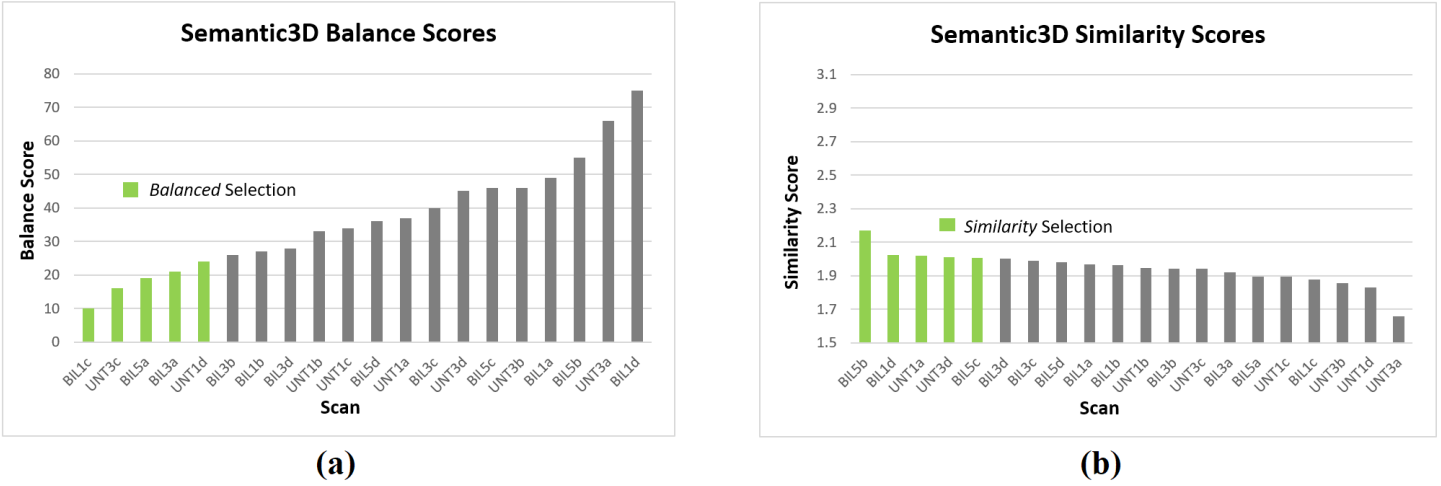


FIGURE 7.25: *Balance* **(a)** and *similarity* **(b)** scores of each scan in the Semantic3D dataset. Green bars denote scans selected by the respective scheme.

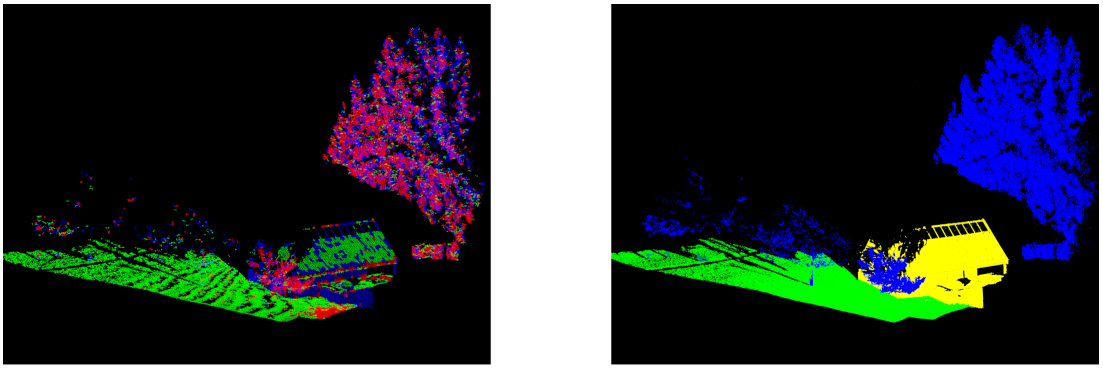
The balance scores (Figure 7.25.a) are considerably more varied than the Montelupo dataset's. The lowest balance score is 10 and the highest is 75, while the scores between them increase linearly. Scans therefore vary from near-perfect balance to high imbalance. Even with general clusters, this means the distribution of supervoxels differs greatly between scans. This is expected given the noticeable variety of scenes the dataset contains.

The similarity scores in Figure 7.25.b mostly lie between 1.8 and 2.0. This is 60 - 66% of the perfect score for the dataset (3, the number of clusters). Although two scans stand out with scores of 2.15 (72%) and 1.65 (55%), most scores are around 10% lower than those from the Montelupo dataset (70 - 76%). This means that scans' supervoxels

are only somewhat similar to the clusters they were assigned to. This is expected given the larger variance between members of general clusters, but still indicates some commonality. It shows that although scenes vary greatly, their objects can be roughly categorised.

The distinct scheme's cluster sizes (from Table 7.14 in the previous example) show two large clusters (size 8 and 9), and three single-scan clusters for Semantic3D. Again, the single scans are chosen either because they are highly distinctive or because k-means is forced to find five non-empty clusters that are maximally different. Given the large variation of scenes in the dataset, it is likely that even a rough description of the scans (the balance and similarity scores derived from general supervoxel clusters) is sufficient for identifying distinct scans.

As in the previous example, scans from each selection scheme are analysed based on their cluster assignments and ground truth. These are provided to analyse why scans are selected and if they are indeed balanced, similar or distinct.

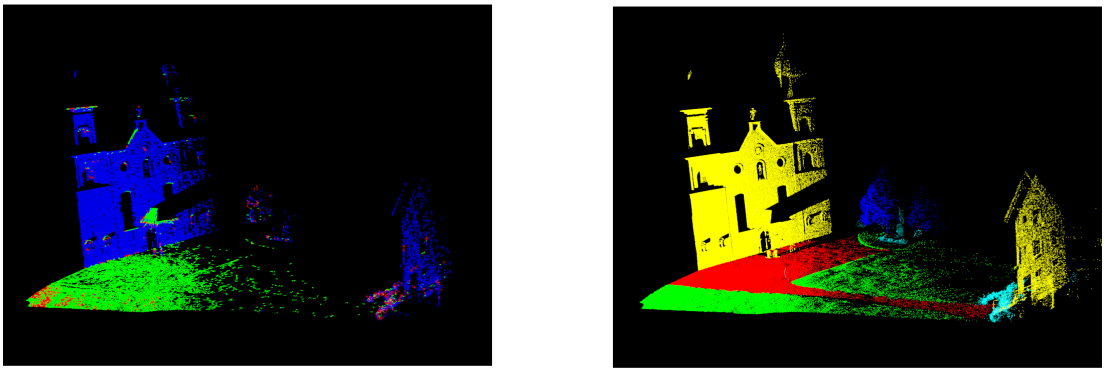


(a) Cluster assignments **(b) Ground truth**

FIGURE 7.26: Exemplar *balanced* scan **BIL1c** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

The balanced scheme achieves an OA of 74%, the highest accuracy of the three schemes. Its selection comprises five scans with balance scores between 10 and 22. The cluster assignments and ground truth of the most balanced scan, BIL1c, are shown in Figures 7.26.a and 7.26.b. The clusters are inconsistent, with many segments from the same structure belonging to different clusters e.g. red and blue segments on the same tree. Despite this, the ground truth has a fairly even distribution of ground, building and vegetation. In terms of class label distribution, the scan is not balanced as it contains only three of the eight classes in the dataset (natural terrain, buildings and high vegetation). Given the inherent imbalance of point classes and imprecise cluster information, it is challenging for the scheme to identify class-balanced scans as it does not have access to

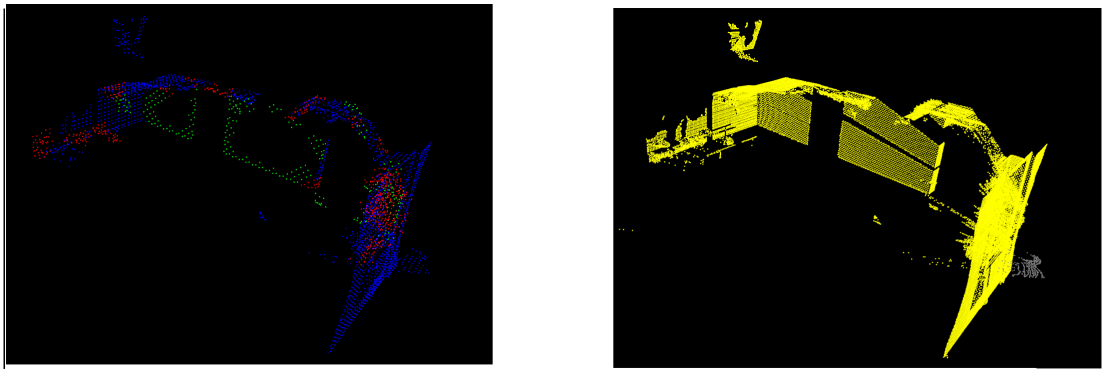
labels. However, by selecting scans with a good distribution of larger estimated classes (clusters), the final selection should cover most point classes.



(a) Cluster assignments **(b) Ground truth**

FIGURE 7.27: Exemplar *similarity* scan **BIL5b** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

The similarity scheme achieves the second highest OA of the schemes with 68%. The exemplar similarity scan, BIL5b, has a similarity score of 2.15 (72% of the perfect score). Its cluster assignments (Figure 7.27.a) and ground truth (Figure 7.27.b) show a high proportion of building and terrain points, well-grouped by the blue and green clusters. The facade and flat terrain have a relatively simple shape compared to other instances of these classes e.g. buildings with roofs and sloped terrain. This could explain why the scan has the highest similarity score; its supervoxels are precisely formed and represent their clusters well. While the ground truth contains five of the eight classes in the dataset, this is not a direct result of the scheme's design. Overall, the scan is not dissimilar to the rest of the campaign and contains structures commonly found in other scans.



(a) Cluster assignments **(b) Ground truth**

FIGURE 7.28: Exemplar *distinct* scan **UNT3a** cluster assignments **(a)** and ground truth **(b)**. Colours in **(a)** denote cluster ID, colours in **(b)** denote class label ID.

The distinct scheme performs the worst of the three on the Semantic3D dataset, achieving an OA of 65%. While this is still 5% higher than the lowest random run, it is 13% lower than the balanced scheme. Figures 7.28.a and 7.28.b show the cluster assignments and ground truth of one of the selected scans, UNT3a. During the scheme's k-means scan clustering, a single-scan cluster is formed with this scan.

Based on the scan's scores, cluster assignments and ground truth, the reasons for this are evident. The scan has the worst similarity score (1.65) and the second worst balance score (65) in the dataset. Inspection reveals it is an "anomaly" scan, a result of the X-coordinate splitting of the dataset. The scan is particularly small, inaccurately clustered and contains almost only building points. The scheme successfully makes a distinct selection, but the selected scan is not descriptive of the dataset and is therefore unsuitable for training. The selection of such training scans explains why this scheme achieves the worst accuracy of the three.

Overall, the varying accuracy of the selection schemes on the Semantic3D dataset is explained by how effectively they utilise scores derived from general, imprecise clusters, as well as how sensibly their underlying principles apply to challenging or problematic datasets. The analysis has shown why, despite these limitations, the schemes can still outperform random selections on inhomogeneous datasets like Montelupo and Semantic3D.

This chapter has explained the experiments used to test the framework's methods, and has given a detailed analysis of the results. By analysing each component separately, this chapter has provided the information needed to answer the research questions and other factors to consider. The next and final chapter concludes the thesis by answering the research questions and discussing possible future work.

Chapter 8

Conclusion

Point cloud classification is a powerful tool that can provide semantic information on points and accelerate time-consuming tasks, but a point classification system's performance greatly depends on the classifier, point feature set and training data that are used. When developing a campaign-based scan classification system, it is important to choose a suitable classifier and feature set, as well as a descriptive set of training scans. These choices are difficult, as a method's success in the literature is not guaranteed to transfer to other contexts, and manually identifying descriptive training scans is a complicated task.

Given the importance and difficulty of these choices, this thesis sought to investigate which classifiers and feature sets performed the best, in terms of speed and accuracy, when applied to datasets of terrestrial laser scans. It also sought to determine how an automated method could select representative, descriptive scans from a campaign, and if a classifier trained on such a selection is more accurate than a classifier trained on randomly selected scans. These goals were formalised as research questions to establish the focus of the thesis. To answer these questions, relevant methods in the literature were reviewed, a framework containing different classification, feature and scan selection methods was developed, and experiments were conducted to evaluate the methods.

The experiments successfully demonstrated the methods' performance and provided the information needed the answer the research questions. In addition, the results gave insight into other factors to consider when developing a point classification system. These answers and insights are discussed below.

8.1 Classifiers and Feature Sets

The first research question, regarding the performance of different classifiers and feature sets, was:

- *Which of a selection of classifiers (Random Forest, Support Vector Machine or Multi-Layer Perceptron), and a variety of point feature sets, achieves the fastest speed and highest accuracy when classifying points from large and diverse terrestrial laser scanning campaigns?*

To address this question, each classifier was combined with three different feature sets and tested on four terrestrial laser scanning campaigns. These three feature sets contained: (1) all extracted features, (2) a selection of features with the highest RF-importance and (3) features with the highest aggregated filter scores.

Classifier Speed
It was found that the MLP was the fastest classifier to train and predict with, while the SVM was the slowest by a large margin. The RF was almost as fast as the MLP to train, but approximately an order of magnitude slower at prediction. However, on the largest dataset, RF prediction required at most 2.5 minutes while the SVM took over 30 minutes. It was also observed that some classifiers are more sensitive to design and parameter choices, and scale better to larger feature vectors, than others in terms of speed.

Classifier Accuracy
The results revealed that the RF was consistently the most accurate classifier on every dataset when paired with the complete feature set. With the selected feature sets, the RF's average overall accuracy (OA) was also higher than the other classifiers using the same features. The SVM and MLP's average OA was only 2-4% lower than the RF, but in some cases was up to 8% lower. Class-based metrics showed that all the classifiers' accuracy varied between classes, but that the MLP and SVM confused classes the most. Lastly, highlighted examples revealed the effect that inconsistent class labels and varying point density can have on classifier accuracy, and how the RF may be more robust to such errors.

Feature Set Speed
It was found that training and prediction with reduced sets of selected feature sets was consistently faster than with the original set. Using the selected sets, the SVM's training and prediction times were reduced by 60-70% and the MLP's were halved. The RF was only 5% faster with the selected sets due to its constant number of trees, but was still

up to four times faster than the SVM. It was also shown that the implemented feature extraction method's speed, 17-31ms per 1000 points, is comparable to others approaches in the literature. Lastly, it was observed that the speed of cylindrical neighbourhood extraction does not scale well with increased radius size, and is particularly slow on datasets with many vertical structures.

Feature Set Accuracy
It was shown that replacing the original feature set with selected features lead to only a small decrease in accuracy. On average, the filter-selected set had 1.4% lower OA than the original set, while the importance-selected set was only 0.2% lower. Where this difference was greater, highlighted examples revealed the descriptive strength of height-based features like vertical range H_{range} when classifying tall structures. Feature scores indicated that 3D shape features like curvature C_λ, omnivariance O_λ and anisotropy A_λ, as well as height features like vertical range H_{range}, height difference ΔH and height deviation σH are the most relevant, a finding that is consistent (where applicable) with the literature. Lastly, the multi-scale features were shown to have some effect on feature scores, with features like verticality V_λ scoring much higher at lower resolutions, but could have had a greater impact if more levels or larger intervals were used.

8.2 Representative Scan Selection

The second research question, regarding the automated selection of descriptive training scans and their comparison to randomly selected scans, was:

- *How do we automatically select a representative set of scans from a scanning campaign that, when used for training, yields a classifier with higher accuracy than a classifier trained on a random selection of scans?*

This question was addressed by designing three scan selection *schemes*, each based on a different concept of what makes a scan representative. The scans selected by these schemes were then compared to random selections by measuring the accuracy of a RF trained on both separately.

Automatic Selection of Representative Scans
Although challenging to evaluate, the results indicated that a method based on segmentation, clustering and scoring can identify a useful set of representative training scans. Visual inspection of scans showed that, in most cases, scans identified by different schemes as *balanced*, *similar* or *distinct* were indeed so, and described other scans in the campaign well. However, the method was revealed to be highly dependent on

the quality of segmentation and clustering output, which requires a careful, campaign-specific choice of parameters and is limited by the use of less accurate methods like k-means clustering. Furthermore, the principles behind the selection schemes were shown to occasionally have negative effects, as seen by the distinct scheme's selection of unique but uninformative scans with very few points. Lastly, although manual selection was not formally tested, the speed of the proposed method (less than 2 seconds per scan) is at least one order of magnitude faster than a conservative estimate of 20 seconds per manually selected scan.

Accuracy of Selected Training Scans

The results showed that the schemes can indeed select scans that yield a more accurate classifier than randomly selected scans. Specifically, the schemes were shown to be the most beneficial on complex and diverse campaigns, where they achieved up to 15% higher overall accuracy than random selection. This reinforces the importance of a representative training set when scans cover large and diverse regions, and demonstrate the success of the schemes in such scenarios. However, it was found that on simple and homogeneous campaigns, the schemes' advantage over random selection is limited. This is to be expected; when there is little variation between the scans in a campaign, changes to the training set have less impact. However, given the schemes' deterministic selection, their advantage over random selection is potentially greater for large campaigns with smaller training sets where random selections are less likely to include descriptive scans. It was also found that no scheme was consistently the best or worst of the three, suggesting that their applicability depends on the campaign. Analysis of exemplar selections indicated this to be the case, with more accurate schemes successfully meeting their scheme's criteria without selecting weakly descriptive scans. Lastly, the method's processing time (at most 35 seconds on tested datasets) is justified by its improved accuracy and consistency over random selection, and scales better to large campaigns where manual selection is more complicated and time-consuming.

8.3 Research Implications and Contribution

This thesis highlights factors to consider when designing a 3D scan campaign classification system. As a classifier, RFs are a sensible choice as they offer high accuracy, relatively low training and prediction times, and are straightforward to implement. However, other classifiers should not be dismissed. An MLP, even with only one hidden layer, achieves a reasonable accuracy but much faster prediction speed than a large RF. An SVM, while much slower, can offer within 1% of an RF's accuracy in some scenarios. Similarly for features, a selected feature set can be almost as accurate as a complete set,

especially when many features are redundant. Furthermore, smaller feature sets greatly decrease training and prediction times, particularly where the classifier does not scale well to larger feature spaces. Lastly, some features are expensive to compute but describe certain classes very well. When choosing a classifier or feature set, one should therefore consider the speed and accuracy requirements of the system, as there is evidently a trade-off between the two.

The research also brings attention to the importance of training scans and how they should be selected. For homogeneous campaigns where scans are very similar, one can randomly choose scans for labelling and training with minimal cost to the classifier's accuracy. However, diverse campaigns require a more sensible approach for the training set to sufficiently describe the rest of the campaign. The research has shown that an automatic solution to this problem is possible. Despite some dependencies, the developed scan selection method has shown that an accurate classifier can be trained on representative, automatically selected training scans. As the method is both automated and deterministic, it should also scale better to larger datasets than manual or random selection, as manually selecting from larger campaigns is laborious, and the chance of randomly selecting descriptive scans from a large pool is lower. The proposed method has therefore set a precedent for future research, which could refine its design and improve its performance.

Overall, this thesis has provided a better understanding of various methods, existing and new, when used as part of a scan campaign classification system. By testing these methods on terrestrial laser scan datasets, including cultural heritage campaigns, the presented methods and findings in this thesis are a valuable resource for other point cloud researchers, as well as cultural heritage preservation groups like the Zamani project. By doing this, this thesis contributes towards important point cloud classification research.

8.4 Future Work

There are a number of possibilities for future work following this research. For example, the proposed scan selection method could be refined. This could involve improving the segmentation and clustering output with more advanced methods, or combining the selection schemes into a single weighted/hybrid method that applies all selection principles simultaneously. Better metrics for measuring how representative a scan is could also be developed, which would provide more information to the schemes, and would enable the statistical evaluation of results to complement visual inspection. More comprehensive testing against manual selection could also be conducted, which may involve lengthy user-training and experiments but would provide a better understanding

of the method's efficacy. Lastly, where scans' GPS coordinates are are recorded for registration, such data could be leveraged to ensure that training sets adequately cover the site's location.

There is also room for future work regarding the use of classifiers and features with point clouds. Other classifiers, or variations of those implemented, could be tested on the same datasets used during this research. These could include classifiers with deeper architectures, such as neural networks with many hidden layers or deep forests [142]. Such classifiers could be tested to determine how well they learn from a small set of labelled scans, as well as the practicality of their training and prediction times. Additional features could be extracted from photographs or range images of the scanning site, followed by an evaluation of their impact on classification or scan selection. With regard to multi-scale features, the effect that varying levels and interval sizes has, and a way to optimise these parameters for a given dataset, could also be investigated.

Appendix A

Source Code

The source code for the framework containing all the designed and implemented methods is publicly available at: https://github.com/pccchr001/3DScanCampaignClassification.

Bibliography

[1] Clustering of Pointclouds into Supervoxels - Theoretical primer. http://pointclouds.org/documentation/tutorials/supervoxel_clustering.php. Accessed: 2019-06-19.

[2] Point Cloud Library (PCL): Module features. http://docs.pointclouds.org/trunk/group__features.html. Accessed: 2019-06-19.

[3] Point Cloud Library (PCL): Module sample consensus. http://docs.pointclouds.org/trunk/group__sample__consensus.html. Accessed: 2019-06-19.

[4] Random Forest Simple Explanation. https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d. Accessed: 2019-06-23.

[5] The Zamani Project. http://www.zamaniproject.org/. Accessed: 2019-06-14.

[6] UNESCO World Heritage List. http://whc.unesco.org/en/list//en/news/1049/en/list/. Accessed: 2019-06-09.

[7] Visual Computing Lab - ISTI - CNR. http://vcg.isti.cnr.it/. Accessed: 2019-10-14.

[8] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[9] Alexander, C., Tansey, K., Kaduk, J., Holland, D., and Tate, N. J. (2011). An approach to classification of airborne laser scanning point cloud data in an urban environment. *International journal of remote sensing*, 32(24):9151–9169.

[10] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. (1999). Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM.

[11] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543.

[12] Barghout, L. and Lee, L. (2004). Perceptual information processing system. US Patent App. 10/618,543.

[13] Bigun, J. (1987). Optimal orientation detection of linear symmetry. pages 433–438. Linköping University Electronic Press.

[14] Bishop, C. M. (2016). *Pattern Recognition and Machine Learning.* Springer-Verlag New York.

[15] Boehler, W., Vicent, M. B., and Marbs, A. (2003). Investigating laser scanner accuracy. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34(Part 5):696–701.

[16] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

[17] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[18] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27.

[19] Chen, D., Zhang, L., Mathiopoulos, P. T., and Huang, X. (2014). A methodology for automated segmentation and reconstruction of urban 3-d buildings from als point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(10):4199–4217.

[20] Chen, J. and Chen, B. (2008). Architectural modeling from sparsely scanned range data. *International Journal of Computer Vision*, 78(2-3):223–236.

[21] Chou, C.-H., Kuo, B.-H., and Chang, F. (2006). The generalized condensed nearest neighbor rule as a data reduction method. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 556–559. IEEE.

[22] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[23] Cracknell, A. P. (2007). *Introduction to remote sensing.* CRC press.

[24] Curtin, R. R., Cline, J. R., Slagle, N. P., March, W. B., Ram, P., Mehta, N. A., and Gray, A. G. (2013). Mlpack: A scalable c++ machine learning library. *Journal of Machine Learning Research*, 14(Mar):801–805.

[25] Dagum, L. and Menon, R. (1998). Openmp: An industry-standard api for shared-memory programming. *Computing in Science & Engineering*, (1):46–55.

[26] Diebel, J. and Thrun, S. (2006). An application of markov random fields to range sensing. In *Advances in neural information processing systems*, pages 291–298.

[27] Dohan, D., Matejek, B., and Funkhouser, T. (2015). Learning hierarchical semantic segmentations of lidar data. In *3D Vision (3DV), 2015 International Conference on*, pages 273–281. IEEE.

[28] Douillard, B., Underwood, J., Vlaskine, V., Quadros, A., and Singh, S. (2014). A pipeline for the segmentation and classification of 3d point clouds. In *Experimental robotics*, pages 585–600. Springer.

[29] Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3:32–57.

[30] El Sayed, A. R., El Chakik, A., Alabboud, H., and Yassine, A. (2018). Efficient 3d point clouds classification for face detection using linear programming and data mining. *The Imaging Science Journal*, 66(1):23–37.

[31] Elseberg, J., Borrmann, D., and Nüchter, A. (2011). Full wave analysis in 3d laser scans for vegetation detection in urban environments. In *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, pages 1–7. IEEE.

[32] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[33] Eum, J., Bae, M., Jeon, J., Lee, H., Oh, S., and Lee, M. (2017). Vehicle detection from airborne lidar point clouds based on a decision tree algorithm with horizontal and vertical features. *Remote Sensing Letters*, 8(5):409–418.

[34] Filin, S. (2002). Surface clustering from airborne laser scanning data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):119–124.

[35] Filin, S. and Pfeifer, N. (2005). Neighborhood systems for airborne laser data. *Photogrammetric Engineering & Remote Sensing*, 71(6):743–755.

[36] Filin, S. and Pfeifer, N. (2006). Segmentation of airborne laser scanning data using a slope adaptive neighborhood. *ISPRS journal of Photogrammetry and Remote Sensing*, 60(2):71–80.

[37] Fischler, M. A. and Bolles, R. C. (1987). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier.

[38] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

[39] Fix, E. and Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, California Univ Berkeley.

[40] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

[41] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

[42] Fu, S. and Desmarais, M. C. (2008). Fast markov blanket discovery algorithm via local learning within single pass. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 96–107. Springer.

[43] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE.

[44] Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58.

[45] Gini, C. (1997). Concentration and dependency ratios. *Rivista di politica economica*, 87:769–792.

[46] Goyer, G. G. and Watson, R. (1963). The laser and its application to meteorology. *Bulletin of the American Meteorological Society*, 44(9):564–570.

[47] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

[48] Gupta, S., Arbeláez, P., Girshick, R., and Malik, J. (2015). Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation. *International Journal of Computer Vision*, 112(2):133–149.

[49] Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017). Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*.

[50] Hackel, T., Wegner, J. D., and Schindler, K. (2016). Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3).

[51] Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato Hamilton.

[52] Hansard, M., Lee, S., Choi, O., and Horaud, R. P. (2012). *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media.

[53] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516.

[54] Haykin, S. and Network, N. (2004). A comprehensive foundation. *Neural networks*, 2(2004):41.

[55] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM.

[56] Jutzi, B. and Gross, H. (2009). Nearest neighbour classification on laser point clouds to gain object structures from buildings. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 1):4–7.

[57] Kaufman, L. and Rousseeuw, P. (1987). *Clustering by means of medoids*. North-Holland.

[58] Kim, S., Hinckley, T., and Briggs, D. (2009). Classifying tree species using structure and spectral data from lidar. In *ASPRS/MAPPS 2009 Specialty Conference*.

[59] Kira, K. and Rendell, L. A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Aaai*, volume 2, pages 129–134.

[60] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282–289.

[61] Lai, K., Bo, L., Ren, X., and Fox, D. (2011). A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE.

[62] Lai, K. and Fox, D. (2010). Object recognition in 3d point clouds using web data and domain adaptation. *The International Journal of Robotics Research*, 29(8):1019–1037.

[63] Landrieu, L. and Simonovsky, M. (2017). Large-scale point cloud semantic segmentation with superpoint graphs. *arXiv preprint arXiv:1711.09869*.

[64] Lari, Z. and Habib, A. (2012). Alternative methodologies for the estimation of local point density index: Moving towards adaptive lidar data processing. *Int. Arch. Photogramm. Remote Sens. Spat. Inform. Sci*, 39:127–132.

[65] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[66] Lee, I. and Schenk, T. (2002). Perceptual organization of 3d surface points. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):193–198.

[67] Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *SIGIR94*, pages 3–12. Springer.

[68] Li, Y., Hu, Z., Cai, Y., and Zhang, W. (2005). Support vector based prototype selection method for nearest neighbor rules. In *International Conference on Natural Computation*, pages 528–535. Springer.

[69] Lior, R. and Maimon, O. (2014). *Data mining with decision trees: theory and applications*, volume 81. World scientific.

[70] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[71] Lodha, S. K., Fitzpatrick, D. M., and Helmbold, D. P. (2007). Aerial lidar data classification using adaboost. In *3-D Digital Imaging and Modeling, 2007. 3DIM'07. Sixth International Conference on*, pages 435–442. IEEE.

[72] Lumini, A. and Nanni, L. (2006). A clustering method for automatic biometric template selection. *Pattern Recognition*, 39(3):495–497.

[73] Ma, S. and Huang, J. (2008). Penalized feature selection and classification in bioinformatics. *Briefings in bioinformatics*, 9(5):392–403.

[74] Marais, P., Dellepiane, M., Cignoni, P., and Scopigno, R. (2019). Semi-automated cleaning of laser scanning campaigns with machine learning. *Journal on Computing and Cultural Heritage (JOCCH)*, 12(3):16.

[75] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

[76] Mollineda, R. A., Ferri, F. J., and Vidal, E. (2002). An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35(12):2771–2782.

[77] Mulder, R. and Marais, P. (2016). Accelerating point cloud cleaning. In *GCH*, pages 211–214.

[78] Munoz, D., Bagnell, J. A., Vandapel, N., and Hebert, M. (2009). Contextual classification with functional max-margin markov networks. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 975–982. IEEE.

[79] Nguyen, A. and Le, B. (2013). 3d point cloud segmentation: A survey. In *Robotics, Automation and Mechatronics (RAM), 2013 6th IEEE Conference on*, pages 225–230. IEEE.

[80] Nguyen, H. T. and Smeulders, A. (2004). Active learning using pre-clustering. In *Proceedings of the twenty-first international conference on Machine learning*, pages 79–86. ACM.

[81] Olvera-López, J. A., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2005). Sequential search for decremental edition. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 280–285. Springer.

[82] Olvera-López, J. A., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2007). Object selection based on clustering and border objects. In *Computer Recognition Systems 2*, pages 27–34. Springer.

[83] Olvera-López, J. A., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2008). Prototype selection via prototype relevance. In *Iberoamerican Congress on Pattern Recognition*, pages 153–160. Springer.

[84] Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., and Kittler, J. (2010). A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143.

[85] Papon, J., Abramov, A., Schoeler, M., and Worgotter, F. (2013). Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2027–2034.

[86] Paredes, R. and Vidal, E. (2000). Weighting prototypes-a new editing approach. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 25–28. IEEE.

[87] Pauly, M., Keiser, R., and Gross, M. (2003). Multi-scale feature extraction on point-sampled surfaces. In *Computer graphics forum*, volume 22, pages 281–289. Wiley Online Library.

[88] Pearson, K. (1896). Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 187:253–318.

[89] Pearson, K. (1900). X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175.

[90] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

[91] Pena, J. M., Nilsson, R., Björkegren, J., and Tegnér, J. (2007). Towards scalable and data efficient learning of markov boundaries. *International Journal of Approximate Reasoning*, 45(2):211–232.

[92] Pfeifer, N., Dorninger, P., Haring, A., and Fan, H. (2007). Investigating terrestrial laser scanning intensity data: quality and functional relations. In *8th Conf. on O3D*, Zurich, Switzerland.

[93] Phillips, P. J., Flynn, P. J., Scruggs, T., Bowyer, K. W., Chang, J., Hoffman, K., Marques, J., Min, J., and Worek, W. (2005). Overview of the face recognition grand challenge. In *Computer vision and pattern recognition, 2005. CVPR 2005. IEEE computer society conference on*, volume 1, pages 947–954. IEEE.

[94] Pu, S. and Vosselman, G. (2006). Automatic extraction of building features from terrestrial laser scanning. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5):25–27.

[95] Pudil, P., Ferri, F. J., Novovicova, J., and Kittler, J. (1994). Floating search methods for feature selection with nonmonotonic criterion functions. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*, volume 2, pages 279–283. IEEE.

[96] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

[97] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

[98] Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

[99] Raicharoen, T. and Lursinsap, C. (2005). A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (poc-nn) algorithm. *Pattern recognition letters*, 26(10):1554–1567.

[100] Reynolds, D. (2015). Gaussian mixture models. *Encyclopedia of biometrics*, pages 827–832.

[101] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE international conference on neural networks*, volume 1993, pages 586–591. San Francisco.

[102] Riquelme, J. C., Aguilar-Ruiz, J. S., and Toro, M. (2003). Finding representative patterns with ordered projections. *Pattern Recognition*, 36(4):1009–1018.

[103] Roy, N. and McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448.

[104] Roynard, X., Deschaud, J.-E., and Goulette, F. (2018). Classification of point cloud scenes with multiscale voxel deep network. *arXiv preprint arXiv:1804.03583*.

[105] Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. (2010). Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE.

[106] Rusu, R. B., Holzbach, A., Blodow, N., and Beetz, M. (2009). Fast geometric point labeling using conditional random fields. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 7–12. IEEE.

[107] Rüther, H., Held, C., Bhurtha, R., Schroeder, R., and Wessels, S. (2012). From point cloud to textured model, the zamani laser scanning pipeline in heritage documentation. *South African Journal of Geomatics*, 1(1):44–59.

[108] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library.

[109] Schoenberg, J. R., Nathan, A., and Campbell, M. (2010). Segmentation of dense range information in complex urban scenes. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2033–2038. IEEE.

[110] Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

[111] Serna, A. and Marcotegui, B. (2014). Detection, segmentation and classification of 3d urban objects using mathematical morphology and supervised learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:243–255.

[112] Settles, B. (2009). Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

[113] Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics.

[114] Settles, B., Craven, M., and Ray, S. (2008). Multiple-instance active learning. In *Advances in neural information processing systems*, pages 1289–1296.

[115] Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM.

[116] Shapiro, L. G. and Stockman, G. C. (2001). *Computer Vision*. Prentice-Hall.

[117] Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer.

[118] Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*, pages 293–301. Elsevier.

[119] Socher, R., Huval, B., Bath, B., Manning, C. D., and Ng, A. Y. (2012). Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*, pages 656–664.

[120] Srisawat, A., Phienthrakul, T., and Kijsirikul, B. (2006). Sv-knnc: An algorithm for improving the efficiency of k-nearest neighbor. In *Pacific Rim International Conference on Artificial Intelligence*, pages 975–979. Springer.

[121] Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston.

[122] UNESCO (1972). Convention concerning the protection of the world cultural and natural heritage.

[123] Vo, A.-V., Truong-Hong, L., Laefer, D. F., and Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100.

[124] Walston, S. (1978). The preservation and conservation of aboriginal and pacific cultural material in australian museums. *ICCM bulletin*, 4(4):9–21.

[125] Wang, Z. and Xue, X. (2014). Multi-class support vector machine. In *Support Vector Machines Applications*, pages 23–48. Springer.

[126] Wang, Z., Zhang, L., Fang, T., Mathiopoulos, P. T., Tong, X., Qu, H., Xiao, Z., Li, F., and Chen, D. (2015). A multiscale and hierarchical feature extraction method for terrestrial laser scanning point cloud classification. *IEEE Transactions on Geoscience and Remote Sensing*, 53(5):2409–2425.

[127] Weinmann, M., Jutzi, B., Hinz, S., and Mallet, C. (2015a). Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:286–304.

[128] Weinmann, M., Jutzi, B., and Mallet, C. (2013). Feature relevance assessment for the semantic interpretation of 3d point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 5:313–318.

[129] Weinmann, M., Jutzi, B., and Mallet, C. (2014). Semantic 3d scene interpretation: a framework combining optimal neighborhood size selection with relevant features. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):181–188.

[130] Weinmann, M., Jutzi, B., and Mallet, C. (2017). Geometric features and their relevance for 3d point cloud classification. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:157–164.

[131] Weinmann, M., Mallet, C., and Brédif, M. (2016). Detection, segmentation and localization of individual trees from mms point cloud data. In *Proceedings of the International Conference on Geographic Object-Based Image Analysis*, pages 1–8, Enschede, The Netherlands.

[132] Weinmann, M., Urban, S., Hinz, S., Jutzi, B., and Mallet, C. (2015b). Distinctive 2d and 3d features for automated large-scale scene analysis in urban areas. *Computers & Graphics*, 49:47–57.

[133] West, K. F., Webb, B. N., Lersch, J. R., Pothier, S., Triscari, J. M., and Iverson, A. E. (2004). Context-driven automated target detection in 3d data. In *Automatic Target Recognition XIV*, volume 5426, pages 133–144. International Society for Optics and Photonics.

[134] Whitney, A. W. (1971). A direct method of nonparametric measurement selection. *IEEE Transactions on Computers*, 100(9):1100–1103.

[135] Wilson, D. R. and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286.

[136] Wright, M. N. and Ziegler, A. (2015). ranger: A fast implementation of random forests for high dimensional data in c++ and r. *arXiv preprint arXiv:1508.04409*.

[137] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920.

[138] Xu, L., Yan, P., and Chang, T. (1988). Best first strategy for feature selection. In *Pattern Recognition, 1988., 9th International Conference on*, pages 706–708. IEEE.

[139] Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863.

[140] Zhang, J., Lin, X., and Ning, X. (2013). Svm-based classification of segmented airborne lidar point clouds in urban areas. *Remote Sensing*, 5(8):3749–3775.

[141] Zhang, R., Candra, S. A., Vetter, K., and Zakhor, A. (2015). Sensor fusion for semantic segmentation of urban scenes. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1850–1857. IEEE.

[142] Zhou, Z.-H. and Feng, J. (2017). Deep forest. *arXiv preprint arXiv:1702.08835*.