ACTIVE LEARNING FOR SEMANTIC LABELLING OF AIRBORNE LIDAR DATA

by

Anna Mewhort
B.Sc., Thompson Rivers University, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Anna Mewhort 2013
SIMON FRASER UNIVERSITY
Fall 2013

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be reproduced without authorization under the conditions for “Fair Dealing.” Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.
APPROVAL

Name: Anna Mewhort

Degree: Master of Science

Title of Thesis: Active Learning for Semantic Labelling of Airborne LIDAR Data

Examining Committee: Dr. Arthur (Ted) Kirkpatrick,
Chair, Associate Professor

Dr. Richard (Hao) Zhang,
Senior Supervisor, Associate Professor

Dr. Suzana Dragicevic,
Supervisor, Professor, Geography

Dr. Shahram Payandeh,
SFU Examiner, Professor, Engineering Science

Date Approved: September 18, 2013
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the non-exclusive, royalty-free right to include a digital copy of this thesis, project or extended essay[s] and associated supplemental files (“Work”) (title[s] below) in Summit, the Institutional Research Repository at SFU. SFU may also make copies of the Work for purposes of a scholarly or research nature; for users of the SFU Library; or in response to a request from another library, or educational institution, on SFU’s own behalf or for one of its users. Distribution may be in any form.

The author has further agreed that SFU may keep more than one copy of the Work for purposes of back-up and security; and that SFU may, without changing the content, translate, if technically possible, the Work to any medium or format for the purpose of preserving the Work and facilitating the exercise of SFU’s rights under this licence.

It is understood that copying, publication, or public performance of the Work for commercial purposes shall not be allowed without the author’s written permission.

While granting the above uses to SFU, the author retains copyright ownership and moral rights in the Work, and may deal with the copyright in the Work in any way consistent with the terms of this licence, including the right to change the Work for subsequent purposes, including editing and publishing the Work in whole or in part, and licensing the content to other parties as the author may desire.

The author represents and warrants that he/she has the right to grant the rights contained in this licence and that the Work does not, to the best of the author’s knowledge, infringe upon anyone’s copyright. The author has obtained written copyright permission, where required, for the use of any third-party copyrighted material contained in the Work. The author represents and warrants that the Work is his/her own original work and that he/she has not previously assigned or relinquished the rights conferred in this licence.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2013
Abstract

Creating training data for classification of airborne LIDAR data is expensive and time-consuming. To label training data, ground truth data is gathered via field surveys or human photo-interpretation of aerial imagery. To avoid getting poor end results due to insufficient training data, organizations often label more training data than is actually needed at a large expense. Using a semi-supervised, active learning approach for both the segmentation and classification of human-made objects and vegetation in urban, airborne LIDAR point clouds, as is proposed in this work, allows a minimal training data set to be created, tested, and expanded in key areas, as-needed, in an interactive, iterative process. The active learning iterations for segmentation gather linkage constraints to apply on the hierarchical clustering. Then, the number of segments is estimated using an enhanced L-method. The active learning iterations for classification gather additional training patches in uncertain areas according to the SVM results.

Keywords: Urban Airborne LIDAR Point Cloud; Active Learning; Semi-Supervised Learning; Constrained Hierarchical Clustering; Segmentation; Support Vector Machine; Classification
To my family, for the love and unconditional support (…and humour) that they always provide through my many adventures, trials, tribulations, and triumphs! Though the work in this thesis is mine, overall, my achievements, apparent sanity, and success are, in reality, a team effort!
“The scientific and technological discoveries that have made war so infinitely more terrible for us are part of the same process that has knit us all so much more closely together.”

“Of all our dreams today there is none more important — or so hard to realise — than that of peace in the world. May we never lose our faith in it or our resolve to do everything that can be done to convert it one day into reality.”

— Lester B. Pearson
Acknowledgements

As my time working on this thesis draws to an end, I would like to thank the many people who have made it possible. Thank you to my senior supervisor, Dr. Hao Zhang, for your guidance, expertise, and patience. Thank you to Mike Parlow, Trevor Hooper, and Joe Kahlert of Object Raku Technologies for the opportunity to make my thesis work into a real-world application. Thank you as well to you and NSERC for the CRD support and opportunity to be a Research Assistant. Thank you to Dr. Suzana Dragicevic, for being on my supervisory committee and for giving me the geography perspective on my work. Thank you to my “lab-mates” at the GrUVi laboratory for your kindness, academic discussions, fascinating and insightful questions about Canadian culture, and fun. Thank you to Simon Fraser University for the Graduate Fellowship. Finally, thank you to my family for the proof-reading, the pixel-counting, and the unconditional support and love throughout my Master’s, my catastrophic knee injury during my Master’s, and always.
# Table of Contents

Approval ii  
Partial Copyright License iii  
Abstract iv  
Dedication v  
Quotation vi  
Acknowledgements vii  
Table of Contents viii  
List of Tables xii  
List of Figures xiii  
List of Acronyms xvi

1 Introduction 1  
   1.1 Contributions ................................................................. 4  
   1.2 Thesis Organization ......................................................... 5  

2 Background 6  
   2.1 LIDAR ................................................................. 6  
      2.1.1 LIDAR Data Collection ............................................. 6  
      2.1.2 Point Cloud Data from Airborne LIDAR ......................... 7
2.1.3 Challenges of Working with LIDAR Point Cloud Data .......... 8
2.1.4 Applications of LIDAR Point Cloud Data .................. 8
2.2 Classification and Labelling of a LIDAR Point Cloud .......... 9
  2.2.1 Feature/Attribute Vector ............................ 9
  2.2.2 Supervised Machine Learning .......................... 10
  2.2.3 Unsupervised Machine Learning ........................ 15
  2.2.4 Semi-Supervised Machine Learning .................... 20

3 Related Work 25
  3.1 Segmentation and Classification of Human-Made Objects in Urban Airborne LIDAR Data .......................... 25
  3.2 Supervised Machine Learning with SVM for Classification of Airborne LIDAR Data .......................... 29
  3.3 Unsupervised Machine Learning with Hierarchical Clustering for Segmentation of Airborne LIDAR Data .............. 32
  3.4 Semi-Supervised Active Learning for Segmentation with Constrained Hierarchical Clustering ....................... 32
    3.4.1 Of Other Remote Sensing Data .......................... 33
    3.4.2 Of Airborne LIDAR Data .............................. 34
  3.5 Semi-Supervised Active Learning for Classification .............. 35
    3.5.1 Of Other Remote Sensing Data .......................... 35
    3.5.2 Of Airborne LIDAR Data .............................. 36

4 Active Learning for Semantic Labelling 38
  4.1 Semi-Supervised, Active Learning for Segmentation and Labelling To Create Training Data .......................... 39
    4.1.1 Hierarchical Clustering .................................. 41
    4.1.2 Selecting the Ideal Number of Clusters with the L-Method ....... 42
    4.1.3 Active Learning to Gather Constraints ..................... 47
    4.1.4 Hierarchical Clustering with Constraints .................. 49
    4.1.5 Enhancement of the L-Method for Constraints ............... 52
    4.1.6 Neighbourhood-Based Attributes Assigned to a Point’s Feature Vector .......................... 57
    4.1.7 Labelling ............................................. 64
  4.2 Semi-Supervised, Active Learning for Classification .......... 64
4.2.1 Training Data Feature Vectors ........................................... 65
4.2.2 Support Vector Machine Classification ................................. 66
4.2.3 Active Learning to Improve Training Data ............................ 70

5 Results ............................................................................. 74
5.1 Test Data ......................................................................... 74
5.2 Software Developed .......................................................... 75
5.3 Performance ..................................................................... 75
5.4 Active Learning Guidance versus Random Selection for Segmentation ........................................... 78
5.5 Active Learning Guidance versus Random Selection for Classification .............................................. 82
5.6 Region-Based versus Point-Based Selection for Active Learning for
Segmentation and Classification ................................................ 85

6 Conclusion ......................................................................... 90
6.1 Future Work ..................................................................... 90

Bibliography ........................................................................ 93

Appendix A Software Requirements and High-Level Design ......... 100
A.1 Requirements .................................................................. 102
A.2 Software Architecture ...................................................... 104

Appendix B Software Detailed Design ...................................... 106
B.1 Template Matching Graphical User Interface ..................... 106
B.1.1 Classes ....................................................................... 113
B.2 Template Creation Attributes and Weights ....................... 115
B.2.1 Classes ....................................................................... 121
B.3 Templates ....................................................................... 121
B.3.1 Classes ....................................................................... 123
B.4 Template Matching Thread Execution Manager ................ 123
B.4.1 Classes ....................................................................... 124
B.5 Template Matching Dynamic-Link Library External Interface .................................................. 124
B.5.1 Interface for Segmentation into Templates ...................... 124
B.5.2 Interface for Classification for Finding Template Matches by Training
and Predicting .................................................................. 129
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.6 LIDAR Point Cloud</td>
<td>132</td>
</tr>
<tr>
<td>B.6.1 Classes</td>
<td>132</td>
</tr>
<tr>
<td>B.7 Support Vector Machine for a LIDAR Point Cloud</td>
<td>134</td>
</tr>
<tr>
<td>B.7.1 Classes</td>
<td>135</td>
</tr>
<tr>
<td>B.8 Utilities</td>
<td>136</td>
</tr>
<tr>
<td>B.8.1 Functions</td>
<td>136</td>
</tr>
</tbody>
</table>
List of Tables

1.1 ASPRS LAS 1.4 File Format Specification Classification Values . . . . . . . . 2

4.1 Default Attributes and Weights . . . . . . . . . . . . . . . . . . . . . . . 63
## List of Figures

1.1 LIDAR Points that were Classified as Vegetation ........................................... 2  
2.1 SVM Decision Boundary Margin Maximization ................................................ 12  
2.2 SVM Non-Linear Kernel Feature Space Transformation ....................................... 13  
2.3 Hierarchical Clustering Tree Example .............................................................. 18  
2.4 Unlabelled Data Points Improving the Decision Boundary Between Labelled  
   Data Points in Semi-Supervised Machine Learning ........................................... 21  
4.1 Selecting the First Geographical Area in which to Develop Training Data ............ 40  
4.2 Graph of the Number of Clusters versus Distance Measures ............................... 43  
4.3 Lines in the Positions of Best Fit on the Graph of the Number of Clusters  
   versus Distance Measures .................................................................................. 44  
4.4 Lines in the Positions of Best Fit with Iterative Refinement ................................. 46  
4.5 Training Patch’s Initial Segmentation ................................................................... 47  
4.6 Hierarchical Clustering Tree Example with Clusters on the Edge of the  
   Decision Boundary Highlighted ......................................................................... 48  
4.7 Graph of the Number of Clusters versus Distance Measures after Hierarchical  
   Clustering with Constraints ............................................................................... 53  
4.8 Skewed Lines in the Positions of Best Fit on the Graph of the Number of  
   Clusters versus Distance Measures .................................................................... 54  
4.9 Skew Corrected with the Enhancement of the L-Method for Constraints ............... 56  
4.10 Training Patch’s Segmentation Improved with Constraints ................................. 57  
4.11 Immediate Neighbours of the Starred Point ...................................................... 59  
4.12 Contiguous Neighbours of the Starred Point ...................................................... 61  
4.13 Training Patch’s Segmentation Improved with Constraints, then Labelled .......... 64
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14</td>
<td>Classification Results with Only One Training Patch</td>
<td>68</td>
</tr>
<tr>
<td>4.15</td>
<td>Probability Visualization of Classification Results with Only One Training Patch</td>
<td>69</td>
</tr>
<tr>
<td>4.16</td>
<td>Misclassification in Areas of Low Confidence</td>
<td>70</td>
</tr>
<tr>
<td>4.17</td>
<td>Classification Results Improved With Five Iterations of Active Learning</td>
<td>72</td>
</tr>
<tr>
<td>4.18</td>
<td>Classification Results Improved With Seven Iterations of Active Learning</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Segmentation Execution Time Versus Number of Constraints</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>SVM Parameter Selection and Training Execution Time Versus Number of Training Data Points</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>Training Patches Improving With Active Learning Using Point-Level Constraints Only</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Chart of Training Patches’ Accuracy Improving With Active Learning Using Point-Level Constraints Only Versus Random Selection</td>
<td>81</td>
</tr>
<tr>
<td>5.5</td>
<td>Classification Results With Iterations of Active Learning</td>
<td>83</td>
</tr>
<tr>
<td>5.6</td>
<td>Classification Results With Active Learning Versus Randomly-Selected Training Patches</td>
<td>84</td>
</tr>
<tr>
<td>5.7</td>
<td>Colours Used for the Classification Results to Label Types of Objects</td>
<td>84</td>
</tr>
<tr>
<td>5.8</td>
<td>Chart of Training Patches’ Accuracy Improving With Active Learning Using Cluster-Level and Point-Level Versus Only Point-Level Constraints</td>
<td>87</td>
</tr>
<tr>
<td>5.9</td>
<td>Training Patch Comparison of the Best Segmentation With Only Point-Level Versus With Both Cluster-Level and Point-Level Constraints</td>
<td>89</td>
</tr>
<tr>
<td>A.1</td>
<td>Main Display of Object Raku Technology’s FTI</td>
<td>101</td>
</tr>
<tr>
<td>A.2</td>
<td>High-Level Interfaces of the New Template Matching Component with the Existing FTI Software</td>
<td>104</td>
</tr>
<tr>
<td>B.1</td>
<td>Template Matching Graphical User Interface for Specifying Parameters for Template Creation</td>
<td>108</td>
</tr>
<tr>
<td>B.2</td>
<td>Template Matching Graphical User Interface Dialogue Box for the Privileged Users for Specifying Parameters for Template Creation</td>
<td>108</td>
</tr>
<tr>
<td>B.3</td>
<td>Template Matching Graphical User Interface for Selecting a Training Patch for Template Creation</td>
<td>109</td>
</tr>
<tr>
<td>B.4</td>
<td>Template Matching Graphical User Interface for the Template Items</td>
<td>110</td>
</tr>
</tbody>
</table>
List of Acronyms

.geoTIFF — TIFF file with georeferencing information embedded in it [RR00]
.las — File Format — binary, maintains three-dimensional LIDAR point cloud data [ASP12]
2.5D Two-and-a-Half-Dimensional
2D Two-Dimensional
3D Three-Dimensional

ACM Association for Computing Machinery
ANN Artificial Neural Network
ASPRS American Society for Photogrammetry and Remote Sensing

CASI Compact Airborne Spectrographic Imager
CPU Central Processing Unit
CRD Collaborative Research and Development, NSERC Grant

DDR3 Double Data Rate Type Three, a type of RAM
DLL Dynamic-Link Library
DSM Digital Surface Model
DTM Digital Terrain Model

FTI Feature Type Interpreter, software program by Object Raku Technology

GB Gigabyte
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoTIFF</td>
<td>File Format — TIFF file with georeferencing information embedded in it [RR00]</td>
</tr>
<tr>
<td>GHz</td>
<td>Gigahertz</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GrUVi</td>
<td>GRaphics, Usability, and VIsnualization, Laboratory Group in the School of Computing Science at SFU</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit</td>
</tr>
<tr>
<td>ISPRS</td>
<td>International Society for Photogrammetry and Remote Sensing</td>
</tr>
<tr>
<td>LADAR</td>
<td>LASER Detection And Ranging, another name for LIDAR</td>
</tr>
<tr>
<td>LAS</td>
<td>File Format — binary, maintains three-dimensional LIDAR point cloud data [ASP12]</td>
</tr>
<tr>
<td>LASER</td>
<td>Light Amplification by Stimulated Emission of Radiation</td>
</tr>
<tr>
<td>LIDAR</td>
<td>Light Detection And Ranging</td>
</tr>
<tr>
<td>MHz</td>
<td>Megahertz</td>
</tr>
<tr>
<td>NSERC</td>
<td>Natural Sciences and Engineering Research Council of Canada</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCS</td>
<td>Projected Co-ordinate System</td>
</tr>
<tr>
<td>pixel</td>
<td>PIcture ELement, the smallest possible point in a digital image</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, Green, Blue, a model to specify colours, may be used in a TIFF file</td>
</tr>
<tr>
<td>RGBA</td>
<td>Red, Green, Blue, Alpha, a model to specify colours, may be used in a TIFF file</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Squared Error</td>
</tr>
</tbody>
</table>
SFU  Simon Fraser University
SPIE  Society of Photo-Optical Instrumentation Engineers
SVM  Support Vector Machine

TBD  To Be Determined
TIFF  Tagged Image File Format [Ado09]

voxel  Volumetric Pixel, the 3D version of a pixel
Chapter 1

Introduction

In recent years, the rapid and widespread collection of large volumes of Light Detection And Ranging (LIDAR) data, via airborne and terrestrial scanners, has created a demand for highly-automated processing of the data and automated segmentation and classification of objects in the resulting three-dimensional (3D) point clouds. In the geospatial information community, the standard practice has been to consider each classified point in the LIDAR point cloud as belonging to one of the classifications of ground/“bare-Earth”, buildings, water, or low, medium, or high vegetation. In fact, the de-facto standard file format for storage and transmission of LIDAR data, LAS, did not support any values of a broader range of classifications than these, other than by the use of a generic one-byte field for any user-defined purpose and value and unstandardized between organizations and pieces of software [ASP12].

As the resolution of collected LIDAR data has become higher, ever more objects are visible in the point clouds, prompting a need for more detailed classifications. In fact, the latest version of the American Society for Photogrammetry and Remote Sensing (ASPRS) LAS file format specification, version 1.4, now explicitly defines more classification values, as listed in Table 1.1, and allows a range of values, 64–255, for user-defined classes [ASP12].

Geospatial software that utilizes the LAS file format can now use the new, more granular range of labels for classification, storage, and transmission with this new standard specification. Using the old specification and its less precise granularity of classification, points that were not part of the ground, buildings, or water classifications often ended up classified as vegetation. This can be seen in Figure 1.1. The cars and other small urban objects are included in the vegetation classification, as visible in the image on the right, showing the
CHAPTER 1. INTRODUCTION

Table 1.1: ASPRS LAS 1.4 File Format Specification Classification Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Created, never classified</td>
</tr>
<tr>
<td>1</td>
<td>Unclassified</td>
</tr>
<tr>
<td>2</td>
<td>Ground</td>
</tr>
<tr>
<td>3</td>
<td>Low Vegetation</td>
</tr>
<tr>
<td>4</td>
<td>Medium Vegetation</td>
</tr>
<tr>
<td>5</td>
<td>High Vegetation</td>
</tr>
<tr>
<td>6</td>
<td>Building</td>
</tr>
<tr>
<td>7</td>
<td>Low Point (noise)</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Water</td>
</tr>
<tr>
<td>10</td>
<td>Rail</td>
</tr>
<tr>
<td>11</td>
<td>Road Surface</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Wire — Guard (Shield)</td>
</tr>
<tr>
<td>14</td>
<td>Wire — Conductor (Phase)</td>
</tr>
<tr>
<td>15</td>
<td>Transmission Tower</td>
</tr>
<tr>
<td>16</td>
<td>Wire-structure Connector (e.g. Insulator)</td>
</tr>
<tr>
<td>17</td>
<td>Bridge Deck</td>
</tr>
<tr>
<td>18</td>
<td>High Noise</td>
</tr>
<tr>
<td>19–63</td>
<td>Reserved</td>
</tr>
<tr>
<td>64–255</td>
<td>User-definable</td>
</tr>
</tbody>
</table>

Figure 1.1: Optical Image (Left) with the LIDAR Points that were Classified as Vegetation Overlayer on It (Right). Note that the cars are classified as vegetation. Data is from Leading Edge Geomatics [Lea13] and the analysis result is from Object Raku Technology’s Feature Type Interpreter [Obj13].
overlay of the classified vegetation points. (The three shades of green correspond to the low, medium, and high elevations of the vegetation classification.)

This thesis work refines the classification of objects that were previously erroneously classified as vegetation.

Because of the large variety of urban objects and the variance of their characteristics in different data sets from different LIDAR scanners that often use different resolutions, a single, broad, automated, one-size-fits-all algorithm for classifications usually produces disappointing results.

To produce a high-quality, segmented, labelled LIDAR data set, many commercial organizations must resort to using expensive human-hours for detailed adjustments to any automated segmentation and classification results.

Though the automated segmentation and classification of objects by computers is a highly developed field, the human aptitude for segmentation and classification of objects still is not matched. The use of human discretion and expertise in geographic information systems for object identification is powerful, yet time-consuming and, hence, expensive. Why not leverage this expertise, improving the efficiency of it in a human/computer partnership, using active learning to enjoy the best of both human expertise and automation?

This thesis proposes an approach for the improved segmentation and classification of human-made objects and vegetation in urban, airborne LIDAR point clouds by collecting and utilizing a number of existing algorithms and concepts from other application domains and adjusting them to be used on airborne LIDAR point cloud data. Software has been developed to fully implement the approach described in this thesis.

It utilizes agglomerative hierarchical clustering with must-link and cannot-link constraints for segmenting a geographic area of the LIDAR point cloud in order to find potential classes of objects for labelling. In the segmentation by clustering, the ideal number of clusters is automatically estimated by utilizing a modified version of the L-method. The segmented and labelled objects can then be used as training data to train a supervised machine learning model [a Support Vector Machine (SVM)] to use for classification on the whole LIDAR point cloud. It utilizes semi-supervised, active learning iterations for both segmentation and classification to elicit and make an efficient use (and re-use) of the human expertise. In the segmentation, semi-supervised, active learning is used for gathering constraints to apply on the clustering to find potential classes of objects for labelling. In the classification, semi-supervised, active learning is used for selecting additional helpful training patches after
viewing the SVM classification result on the whole LIDAR point cloud. This approach and the related algorithms will be explained in more detail in Chapter 4.

1.1 Contributions

The approach outlined in this thesis makes the following novel contributions:

- **Semi-supervised, active learning for segmentation and labelling of airborne LIDAR data:** This thesis uses semi-supervised, active learning iterations for the segmentation and labelling of airborne LIDAR data (using constrained hierarchical clustering).

- **Semi-supervised, active learning for classification of airborne LIDAR data:** This thesis uses semi-supervised, active learning iterations for the classification of airborne LIDAR data without the added information of hyperspectral data.

- **Two types of semi-supervised, active learning embedded in each other’s iterations on remote sensing data:** This thesis utilizes two levels of active learning — one set of active learning iterations for the segmentation and labelling of airborne LIDAR data to make training data and another set of active learning iterations for classification with a SVM. After utilizing all of the labelled training data areas to train a SVM and performing a prediction over a larger geographic area, new training areas (with their own sub-iterations of active learning) can be added in key geographical areas that have high uncertainty or exhibit errors. Alternatively, the previous areas can be removed or improved and the SVM training and prediction can then be re-run.

- **L-method enhancement for constrained hierarchical clustering:** A modified version of the L-method for selecting the number of clusters was developed to work with constrained hierarchical clustering instead of with the standard hierarchical clustering on which the original algorithm was defined.

- **Batch selection of LIDAR data points in both active learning techniques:** Similar to the approach for hyperspectral data that was described by Stumpf, Lachiche, Malet, Kerle, and Puissant [Stu+12], this thesis uses a novel, application-specific enhancement to the batch selection of points in both of the semi-supervised techniques. Using small geographic regions instead of single points or a set of points in disparate
areas, efficiency can be increased in regards to total time (human and computing), allowing the user to infer context and label sets of points according to that context. The constrained hierarchical clustering allows users to specify whole regions of points that are must-link or cannot link. The semi-supervised iterations with adding training data and re-running the SVM allow the user to define new training data in small, geographical “training patches”.

- **Soft segmentation:** This thesis work uses a “soft segmentation”. It uses the power of neighbourhood-based attributes without “locking” a point into that neighbourhood segment before the main clustering for segmentation. This allows some freedom for a point to move between segments if some other attributes are stronger. (If the initial segmentation defining the neighbourhoods is incorrect, a point could still “break-out” if it really does not fit based on other attributes.) In addition, the clustering for segmentation is only done to simplify the labelling of the points to create the training data. The data points with their labels are used to train the SVM individually. Each data point may have some attributes that are derived from its neighbourhood, but the SVM does its classification point-by-point. This eliminates some of the worry of under or over-segmenting and hence, misrepresenting the composite points.

To the best of the author’s knowledge, this is the first work that has applied these techniques to this type of remote sensing data.

### 1.2 Thesis Organization

Chapter 2 explains the technical background of the theories, algorithms, technologies, and data that are used in this thesis work. Chapter 3 lists other academic works that contribute to or overlap with the work in this thesis. Chapter 4 presents the details of the algorithms and contributions of this thesis, along with the reasons that this approach was taken. Chapter 5 exhibits some qualitative and quantitative results of the technique presented in this thesis. Concluding thoughts and ideas for future work are given in Chapter 6.

Appendix A outlines the requirements and the high-level design for the new software that was written to implement the techniques outlined in this thesis. Appendix B goes into more details of the design of the modules of the new software. These appendices are fairly detailed and may be skipped without loss of understanding in any of the chapters.
Chapter 2

Background

The following sections are a high-level overview of the theories, technologies, and data that are utilized in this thesis.

2.1 LIDAR

The data on which this thesis work focusses is LIDAR point cloud data.

2.1.1 LIDAR Data Collection

LIDAR data is collected by using an active sensor. The active sensor transmits a beam of LASER light and then receives the backscattered reflection of the beam after it has contacted and then reflected from an object. The LASER light is typically in the visible or near-infra-red part of electromagnetic spectrum.

To be used as a remote sensing instrument, a LIDAR active sensor is paired with highly-accurate timers, and, if it is mounted on a mobile base (such as a vehicle or an aircraft), it is also paired with devices to record the location and position of the sensor. This is typically a Global Positioning System (GPS) to record exact locations and, on an aircraft, an Inertial Measurement Unit (IMU) to exactly record the roll, pitch, and yaw of the aircraft. Using this precise location and time duration of the transmission and receipt of a pulse, the range to the contacted object can be determined. In the case of a LIDAR sensor on an aircraft, the range, position, and angle of transmission are used to calculate the location and elevation.
of the point on the object that was contacted by the pulse on the ground. The LIDAR data that is used in this thesis is airborne LIDAR data, taken from an aircraft.

Often, part of a transmitted pulse is reflected back to the sensor at a certain range, but another part of the transmitted pulse goes further and is reflected from a further range. In the case of airborne LIDAR, part of the pulse may first be reflected from the leaf of a tree, but another part of the pulse goes further and is reflected from the ground. In this case, the sensor receives returns from multiple different ranges from a single transmitted pulse.

The capabilities and resolution of LIDAR systems vary greatly.

Some LIDAR sensors record discrete values for the intensity of reflected pulses, and some record the full shape of the waveform returned. The LIDAR data used in this thesis has discrete values for the intensity but not the full waveform. Using full waveform data could be an area of enhancement for future work, as detailed in Section 6.1.

The huge volumes of data from multiple scanning passes are filtered, combined, and registered with each other to form a mosaicked 3D model. With the accuracy of the timing, GPS, and IMU, the accuracy of an airborne LIDAR system can rival aerial photogrammetry. [CW11]

2.1.2 Point Cloud Data from Airborne LIDAR

Each pulse, after being reflected, returned to the sensor, and processed, results in a data point with an x and a y value, and one or more z-values — one from each return, possibly at different z-value elevations. In addition to these spatial values, there are also the values for the intensity of the reflected pulses. The de-facto standard file format for storage and transmission of these LIDAR point clouds is the .las file format, defined by the ASPRS.

Even if the LIDAR active sensor transmits pulses in a regular timing, the calculated positions on the ground of the objects contacted is irregularly spaced because of differing elevations and directions of aircraft flying patterns, etc.

For many applications of LIDAR data, a regularly-spaced grid of the data points is preferred for simplicity, aesthetics, and speed of processing. Hence, LIDAR point clouds are often, but not always, interpolated and re-sampled to be made regularly-spaced.

The algorithms in this thesis make no assumptions that the LIDAR data has been re-spaced in a grid format; however, the LIDAR data that was available for testing in this thesis work has indeed already been interpolated to be regularly-spaced. Instead of being
stored in .las files, this type of data can be stored in the .geoTIFF file format, following a regular grid.

### 2.1.3 Challenges of Working with LIDAR Point Cloud Data

A major challenge in working with LIDAR data is the volume of data. As outlined by Schön, Mosa, Laefer, and Bertolotto [Sch+13], using data from Laefer and Pradhan [LP06] and Hinks, Carr, and Laefer [HCL09], low-density LIDAR data has at least 30 points per metre squared and modern high-resolution LIDAR data can have 225 points per metre squared. That is 225 000 000 points for only one kilometre squared. Using LIDAR data for urban planning across a city requires many multiples of this.

This large volume of data cannot be all held in computer memory at once, and, as such, any software that processes LIDAR data needs to be able to work with it in smaller batches, such as with tiles or with a buffering system. It is hard to leverage full contextual information when only small pieces of the data are considered, as noted by Lewis, McElhinney, and McCarthy [LMM12].

Modern database management systems do not yet have full support for true 3D data storage, querying, and analysis [Sch+09]. In the current software that was developed for this thesis, no database management system was used. In future work, as listed in Section 6.1, the speed of point cloud access could be expedited with a true 3D database implementation with indexing, as outlined by Schön, Mosa, Laefer, and Bertolotto [Sch+13] and Lewis, McElhinney, and McCarthy [LMM12].

### 2.1.4 Applications of LIDAR Point Cloud Data

There are many real-world applications for LIDAR for monitoring and measuring the physical world.

One of the most common uses for LIDAR data is to make a model that is often also a precursor to other analyses and applications of LIDAR data: a Digital Terrain Model (DTM). A DTM is also sometimes called a bare-Earth model. It is a model of the ground without any buildings or vegetation on it. With the capability of LIDAR to detect returns from both the tops of trees and the ground below it, for example, it is possible to filter higher points and make a DTM from the lower (and last) return pulse points. The DTM’s must typically be established before doing any other type of analysis, so that there is a
“ground” baseline to work from in determining which other points are part of vegetation or buildings, etc. There are many research papers and techniques focussed on finding a good DTM, as its quality affects the quality of all other analyses that use it as the baseline for “ground”. Meng, Currit, and Zhao [MCZ10] and Sithole and Vosselman [SV04] give a good survey of research on DTM creation.

LIDAR data is used to detect buildings, roads, and vegetation for urban, highway, pipeline, and wireless communication network planning. Floodplain mapping and forest canopy analyses for calculating the forest biomass both use LIDAR data, as well. LIDAR data can be used to monitor erosion changes to a coastline and detect the changes of a glacier over time. [CW11; MK11]

For most of these applications, some analyses need to be conducted on the LIDAR data point cloud. First, the DTM of the geographical area needs to be found. Then, the contents of the LIDAR point cloud must be determined. Which points are part of a building? Which ones reflected off of a tree canopy? Which points are cars? The points in the point cloud need to be classified. For some applications, this is the analysis that must be done — the intent is to determine the objects in the LIDAR point cloud. For other applications, this is a precursor to performing other higher-level analyses, such as temporal change detection. This thesis work focusses on distinguishing human-made objects from vegetation in urban environments, after the DTM and buildings have been correctly identified.

### 2.2 Classification and Labelling of a LIDAR Point Cloud

Classification of the items in a two-dimensional (2D) image involves assigning each pixel to classes of objects. Classification of a 3D LIDAR point cloud is a 3D extension of this concept.

#### 2.2.1 Feature/Attribute Vector

A point must be assigned to a class of objects based on its characteristics — point-based characteristics, such as its spatial location or intensity; or characteristics that are derived from it and its immediate neighbours, such as normal direction, texture/smoothness, ranges, and averages; or characteristics that are derived from the other points that are likely contiguous, all part of one continuous object, such as shape, number of points, and dimensions; or characteristics from points even further away that could suggest a particular
context, such as a point that is likely on a boat being within an area that is likely water or a point that is likely on a car being situated (parked) on the side of a set of points that is likely a road. This thesis uses characteristics, also called attributes or features, of a point that are point-based, immediate-neighbour-based, and contiguous-neighbour-based. The addition of attributes/features that are based on context is left for future work, as listed in Section 6.1.

The set of chosen characteristics that are used to decide to which class of objects that a point belongs is called the attribute or feature vector for that point. The right set of attributes/features must be chosen that characterize a point into the right class. Including useless attributes increases computation time without increasing the quality of the classification. Including attributes that are redundant in distinguishing the class in the classification could skew the importance/magnitude of those attributes in comparison to other non-redundant attributes.

The range of values that each attribute/feature in the feature vector can take in the data is called its “feature space”. The feature space is often normalized for each attribute so that different offsets and ranges on different attributes do not arbitrarily skew the importance of one attribute versus another.

The importance of one attribute versus another can be intentionally increased or decreased, however, based on its power in distinguishing one class from another. A weighting vector that parallels the feature vector can be used for this.

2.2.2 Supervised Machine Learning

Supervised machine learning is often used for LIDAR point cloud classification. Supervised machine learning is a process in which a computer model is “trained” by using examples of the input and the expected output. The parameters that define a model are adjusted to fit each training data example to produce the desired output. In the case of using supervised machine learning for classification, the input training examples are the attribute/feature vectors of a set of points, and the expected outputs are the proper object class labels that should be assigned to each of those points. These inputs and desired outputs are the “training data”.

Before the training phase is ever executed on a machine learning model, the training data must first be gathered, labelled, and verified. A label must be set for each intended output for each input of the training data. This is the “supervised” aspect of “supervised machine learning”.
After the training phase, the machine learning model has been tuned to predict outputs from inputs. “Decision boundaries” have been formed in the model, distinguishing which inputs will produce which outputs. With a new input, a new feature vector, the model will predict the output, in this case, the object classification.

There are many algorithms for classification using supervised machine learning. They vary in the underlying adaptive model that is adjusted to fit the training data. There are parametric (training data is discarded after the model is trained) algorithms, such as discriminant functions like Fisher’s, discriminative models like logistic regression and Artificial Neural Networks (ANNs), and generative models like Naive Bayes.

There are non-parametric (all or a subset of the training data are kept and used for predictions), discriminative models such as K-Nearest Neighbour and SVM’s.

There are also combination models that combine the models listed above in committees like Bagging, Boosting, Decision Trees, and Mixtures of Experts [Bis07].

This thesis work uses a SVM with a Radial Basis Function (RBF) kernel for the supervised machine learning model for classification based on its success in other airborne LIDAR research. [Lod+06; SZ07; Li+07; AM08; DBG08; Koe+08; MSB08; Fra+10; TS11]

**Support Vector Machine**

The modern (soft-margin) version of the SVM was introduced by Cortes and Vapnik [CV95] in 1995. It is a supervised machine learning model that, during training, tries to find a decision boundary between two classes that maximizes a margin between the decision boundary and any of the data samples. Figure 2.1 shows a simple example of this with two feature dimensions, \( x_1 \) and \( x_2 \). The decision boundary labelled \( H_1 \) would not split the classes correctly. The decision boundary labelled \( H_2 \) would split the classes correctly but just barely. The decision boundary labelled \( H_3 \) separates the classes with the maximum margin, measured perpendicularly to the decision boundary.

Finding the decision boundary between two classes that maximizes a margin between the decision boundary of the basis function and any of the data samples is a constrained optimization problem, solved with Lagrange multipliers. This solution can be reformulated into its dual representation as a kernel function, where the optimization is done on a function of the sample points instead of on the basis function.

The kernel function could be linear or non-linear. If it is non-linear, such as the RBF kernel, then the data sample points are transformed by the kernel into a higher-dimensional feature space.
Figure 2.1: SVM Decision Boundary Margin Maximization, by User:ZackWeinberg, based on PNG version by User:Cyc, 2012, at http://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg. Released under Creative Commons Attribution-Share-Alike License 3.0 (CC-BY-SA-3.0) (http://creativecommons.org/licenses/by-sa/3.0), via Wikimedia Commons
Hence, the decision boundary is then actually a hyperplane between the transformed feature values of data samples. Figure 2.2 depicts sample points’ feature vectors in the original feature space, where the classes are not linearly separated, and also transformed into a higher-dimensional feature space by a non-linear kernel, where the classes are then linearly separable.

Hence, unlike many other models, SVM’s are able to classify data that is not linearly-separable in the original feature space, with the selection of the right kernel for the data set.

Once a SVM is trained, that is, the placement of the decision boundary hyperplane has been selected based on the data points, then only the data points that are on the margin’s edge, defining the placement of the decision boundary hyperplane need to be kept. They are the only sample points that are used in predictions. They are called the Support Vectors.

The trained model evaluates the new data points with the tuned kernel and the Support Vectors, resulting in a positive or negative value, indicating one of two classes (positive or negative).
Some common kernels are linear, polynomial, RBF, and sigmoid. This thesis work uses the RBF kernel, as many researchers, such as Lodha, Kreps, Helmbold, and Fitzpatrick [Lod+06] and Trinder and Salah [TS11], have shown it to be effective for remote sensing data, including LIDAR. The RBF kernel has one configurable parameter, $\gamma$, that controls its shape (related to the variance of a Gaussian) and needs to be picked based on the training data. It is usually found empirically by methodical testing with the training data set.

If the classes overlap and their feature vectors are not separable, even in the high-dimensional space, SVM’s allow for some slack to allow the hyperplane still to be defined, but to have a soft margin, allowing some training data to be misclassified. If a training point is on the wrong side of the decision boundary, it contributes a penalty term to the optimization that increases with the distance from the decision boundary. The strength of the penalty term, $C$, is configurable and must be supplied by the user of the SVM, based on the training data. It is usually found empirically by methodical testing with the training data set.

A single SVM is a binary classifier, that is, for only two classes; however, multiple SVM’s can be combined in order to classify multiple classes. Multi-class SVM’s have been constructed in two different ways — one-versus-the-rest and one-against-one. In the one-versus-the-rest version, one SVM is trained for each class. The training data from that class is used as the positive examples, and the training data from all of the other classes are used as the negative examples. In the one-versus-one version, a SVM is trained for each combination of classes. The training data from one class is used as the positive examples, and the training data from another class is used as the negative examples. If there are $K$ classes, then, this requires $K(K - 1)/2$ SVM’s to be trained. For both methods, a voting scheme between the multiple SVM’s is used to decide on the classification. [Bis07; CL11; Wik04]

**Training Data**

The collection of training data for supervised machine learning is a non-trivial process, often being very time-consuming and expensive. It is also hard to know what are “enough” examples for each class, so often users over-estimate and over-produce the quantity of training data to ensure that there is not an inadequate amount for the supervised training to be successful. Even with a large quantity of training data, it is very possible that not all “key” areas get selected for training data. There may be classes not present in the training areas that exist in the larger area. [CW11]
In addition, training data created on one data set may not be applicable to a similar data set, even if the same type of LIDAR scanner is used, with the same scanning height and sampling frequency. For example, the trees in a LIDAR data set taken in the summer may look very different from the same trees in the winter. If different scanners are used, there is an even lower probability that the training data will be able to be reused for accurate effects. A car or a tree will have quite different characteristic attributes at different resolutions.

2.2.3 Unsupervised Machine Learning

There is another type of machine learning that does not use examples of the input and their expected output. In unsupervised machine learning, there is no training data — no correctly-labelled outputs that are to be the result from a set of inputs. There is no supervision on being given the “correct” outputs. There are only the input examples — the attribute/feature vectors of a set of points but no expected output class labels for them.

One of the types of unsupervised machine learning, called clustering, tries to find homogeneity among the input data, the feature vectors. If natural clusters in the data can be found, these groups make likely candidates for segmentation into possible classes. Clustering is sometimes called “unsupervised classification” in remote sensing literature, as the clustering segments data points into classes, even though there are no labels assigned to those classes.

There are many algorithms for clustering, as well. They vary in how the clusters are found and developed and the criteria that is used for determining which points should be “homogeneous-enough” to be in a cluster.

k-Means Clustering

One popular clustering algorithm is the k-means algorithm. In k-means clustering, seed points are chosen, usually randomly, to be the cluster centres. Every other data point is then placed in the cluster with the cluster centre to which it is the closest. After all of the data points have been placed in clusters, the cluster centres are re-calculated to be the mean value of the points that have been placed in that cluster. This may move the centre point. Then, each point is again placed in the cluster with the centre point to which it is closest. These steps are performed iteratively until the cluster centres no longer move between iterations. For placing a data point into the cluster with the closest
centre point, the distance between the data point and the cluster centre must be calculated. In the k-means algorithm, the Euclidean distance is used as the distance metric. (The Euclidean distance is found by using the Pythagorean formula — taking the difference between each attribute value in the feature vector of the data point and the feature vector of the cluster centre, then squaring each one, summing them all, and then taking the square root: $\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_f - b_f)^2}$, where $a_1, ..., a_f$ are the attribute values in the feature vector of the cluster centre and $b_1, ..., b_f$ are the attribute values in the feature vector of the data point, and $f$ is the count of the number of attributes in the feature vectors.)

There are many variations of this algorithm. For example, k-medians clustering uses the median to calculate the new centre points instead of the mean; k-medoids clustering requires the centre points to be actual existing data points and uses Manhattan distances: $|a_1 - b_1| + |a_2 - b_2| + \ldots + |a_f - b_f|$ instead of Euclidean distances as the distance metric; expectation-maximization is a generalization of the algorithm that uses a Gaussian distribution instead of the mean and assigns probabilities to cluster assignments.

These k-means types of algorithms are non-deterministic. The movement of the cluster centres could stop at a local minimum instead of a global minimum. If the seeds for the cluster centres are selected randomly, then different local minimums may be found on different executions of the algorithm. For these reasons, the k-means type of algorithm needs to be run many times with a variety of starting seeds as the cluster centres in order to find the global minimum for the clustering. Even then, there is no guarantee that the minimum found is indeed the global minimum and not just the best local minimum so far.

Knowing the number of clusters into which to segment the data is a key input to these k-means types of algorithms. The number of seed points to use for cluster centres relies on this key piece of information to start the algorithm. In a data set, this may not be known. There are, however, variations of the algorithm that will increase the number of clusters based on a distance threshold, considering intra-cluster distances and inter-cluster distances to determine if thresholds have been exceeded and more clusters are needed. For these variations, though, instead of needing to know the number of clusters, the thresholds need to be known in advance. These may not be known for a data set either. Many implementations try to determine these values — either the number of clusters or the threshold values — empirically, by trying multiple iterations of the k-means type of algorithm with different values. [Hoo+04; Bis07; Can10; Eve+11; MK11]
The fact that k-means clustering requires many iterations to hopefully, but not definitely, find the global minimum and the fact that k-means clustering requires many iterations to test potential values for the number of clusters, if this is not known in advance, proved to be prohibitive in utilizing this algorithm for this thesis work. Early versions of this thesis work used k-means clustering to segment unlabelled areas of the LIDAR point cloud, but, the ideal number of clusters was not pre-known and a global minimum was desired. Instead, it was replaced with hierarchical clustering, which is much slower than a single iteration of k-means, but requires only one iteration/execution, so had a feasible speed for this application in comparison to the many iterations of k-means that were needed for confidence in a global minimum and for determining the number of clusters to use. Hierarchical clustering is a deterministic algorithm that allows the selection of the number of clusters as one of the last steps, instead of the first, as is needed in k-means.

Hierarchical Clustering

Hierarchical clustering starts by finding the distance from every data point to every other data point, by using the attribute values in each data points’ feature vector to calculate a distance metric. Two common distance metrics are the Euclidean distance or the Manhattan distance, although there are also many others. After all of the distances are calculated between the data points, a hierarchy of clusters is created. One version of the algorithm does this in a top-down fashion, and another proceeds from the bottom-up. They work similarly, except that the top-down one divides clusters, and the bottom-up one agglomerates clusters. In the agglomerative, bottom-up version, at the start, each data point is considered to be a cluster. If there are \( n \) data points, there are \( n \) clusters. Then, the two data points that are the closest, i.e. they have the smallest distance between them, are joined to make a cluster. Then, there are \( n - 1 \) clusters. New distance metrics are calculated from the newly-formed cluster to the other data points. Now, the two clusters with the next smallest distance between them are merged. There are then only \( n - 2 \) clusters. The clusters are merged, one by one, until there is only one cluster. This process can be represented by a tree, with each level representing a certain number of clusters; each level of the hierarchy of the tree has one more or one less number of clusters than the ones above or below it. Figure 2.3 shows an example of a tree resulting from a hierarchical clustering of six data points named \( a \) through \( f \). The formation of this hierarchy is completely deterministic, determined by the distances between the feature vectors of each data point and the distance metric used to calculate the distances between them.
Figure 2.3: Hierarchical Clustering Tree Example
After the hierarchy with number-of-cluster values from \( n \) to one is complete, then the segmentation for a particular number of clusters can be found by cutting the tree at the level that has that number of clusters. This is shown in Figure 2.3 for cutting the tree at the level with four clusters.

The distance calculations between two clusters can be done in a variety of ways. One popular way is to take the pairwise distances between each data point in one cluster with each point in the other cluster and average them, which is called average-linkage. Another option is to take the pairwise distances between each data point in one cluster with each point in the other cluster and then, use the smallest one (shortest distance) to represent the distance between the two clusters. This use of the closest neighbour is called single-linkage. A third popular option is to take the pairwise distances between each data point in one cluster with each point in the other cluster and then use the largest one (furthest distance) to represent the distance between the two clusters. This use of the farthest neighbour is called complete-linkage or maximum-linkage.

An advantage of hierarchical clustering is that the distances and various clustering structures for different numbers of clusters are all computed in the single execution. If a different number of clusters needs to be tried, then the tree can just be re-cut at a different level to get the clustering for that number of clusters. Alternatively, the distances that are calculated for all of the points and clusters can be analysed to look at the increasing distances between clusters in the order in which they are combined to try to determine the ideal number of clusters. [Hoo+04; Can10; Eve+11; MK11]

This thesis work uses agglomerative hierarchical clustering with a Euclidean distance metric and complete-linkage for segmenting an unlabelled area of the LIDAR point cloud in order to find potential classes of objects for labelling, which then can become training data for training a supervised machine learning model (a SVM) for application to the whole LIDAR point cloud.

**Hybrid Clustering**

There are also many clustering algorithms that are hybrids of other clustering algorithms. They attempt to compensate for the weaknesses of the constituent algorithms and capitalize on their strengths by combining them. For example, there is a hierarchical k-means algorithm which builds a tree hierarchy from the top-down by using k-means to bisect a cluster into two at each level of the tree hierarchy.
2.2.4 Semi-Supervised Machine Learning

Semi-supervised machine learning is not completely supervised nor completely unsupervised. As its name implies, semi-supervised machine learning is a blend of both supervised and unsupervised machine learning paradigms.

It uses input training data (the attribute/feature vectors of a set of points) that has no correctly-labelled outputs, as is done in the unsupervised machine learning paradigm. It also has some added information though — a supervised component, in an attempt to improve the predictions.

One of the most common types of semi-supervised machine learning is semi-supervised classification. In this type of semi-supervised machine learning, the added supervised information is a set of expected outputs for some of the input data points. Hence, some of the input data points have labels and some do not. The labelled and unlabelled data points are both used in training the machine learning model. Figure 2.4 illustrates a simple example where the unlabelled data could help in training the model better than just the two labelled data points could in illustrating the distribution, density, and separation of the input data points.

This type of semi-supervised machine learning for classification is essentially a supervised classification that uses some additional unlabelled (unsupervised) data for some additional information.

Another type of semi-supervised machine learning is constrained clustering.

Constrained Clustering

In this type of semi-supervised machine learning, the added supervised information is a set of constraints specifying that some points should be in the same cluster/class and that some should not be.

- The must-link constraints specify that a set of input data points should be in the same output cluster/class.
- The cannot-link constraints specify input data points that should NOT be in the same output cluster/class.

This type of semi-supervised machine learning for clustering is essentially an unsupervised clustering that uses some additional (supervised) data in the form of constraints for some additional information. [WC00; CSZ06; ZG09]
Figure 2.4: Unlabelled Data Points Improving the Decision Boundary Between Labelled Data Points in Semi-Supervised Machine Learning, by Techerin (Own work), 2012, at http://commons.wikimedia.org/wiki/File:Example_of_unlabeled_data_in_semisupervised_learning.png. Released under Creative Commons Attribution-Share-Alike License 3.0 (CC-BY-SA-3.0) (http://creativecommons.org/licenses/by-sa/3.0), via Wikimedia Commons
There are many algorithms for constrained clustering, utilizing the full scope of algorithms that exist for unsupervised clustering, with modifications to the algorithms to consider the must-link and cannot-link constraints in determining the resulting clustering.

The must-link and cannot-link constraints can be specified for pairs of points or for larger sets, such as a whole cluster. There are also variants in which the must-link and cannot-link constraints are actually soft constraints — there is only a penalty for violating the constraints, but violating them is not an impossibility. There are also variants where there are other types of constraints, such as threshold constraints on within and without cluster distances, called a $\delta$-constraint and an $\epsilon$-constraint [DR09]. There are variants where the must-link and cannot-link constraints are additionally used for adjusting the distance metric calculations by applying weights to the different attributes in the feature vectors to encourage the clustering that is defined by the must-link and cannot-link constraints. [BDW08]

This thesis work uses a modified version of agglomerative hierarchical clustering that takes into account must-link and cannot-link constraints. After the distances are calculated between each data point with every other data point, the distances are adjusted based on the constraints. For data points that have a must-link constraint together, the distance between them is set to zero. For data points that have a cannot-link constraint between them, the distance between them is set to a very large value to represent infinity (but within computing numeric capabilities). This facilitates the points that have a distance metric of zero to be clustered together and discourages points with a very large distance metric from being clustered together. All of the points that have a distance metric of zero between them will be the first ones to be combined in the agglomerative hierarchical clustering algorithm, and the points that have a very large distance metric between them will be the very last to get combined, hence, using slightly soft constraints. The must-link and cannot-link constraints can be violated in order to select a certain number of clusters at a certain level of the hierarchy, but, it is only at the top and bottom extremes of the hierarchy, and the distance metrics on these agglomerations reflect this fact, with a zero-value distance metric on the agglomerations that are must-link and a very high penalty on the cannot-link agglomerations. The technique used in this thesis work for selecting the ideal number of clusters avoids selecting a number of clusters in these levels of the hierarchy (that would violate these slightly-soft constraints), as detailed in Chapter 4. This is a slight modification to the technique outlined by Klein, Kamvar, and Manning [KKM02], but now applied to LIDAR data, as described in more details in Chapter 3 and Chapter 4.
CHAPTER 2. BACKGROUND

Using the must-link and cannot-link constraints additionally for guiding the adjustment of the distance metric calculations by applying weights to the different features in the feature vectors is an area for future work, as listed in Section 6.1.

User Interaction/Feedback

Many frameworks for semi-supervised learning elicit decisions from a human user in order to gather the added, supervised information that is used along with the unsupervised information to make the predictions. In semi-supervised classification, a user may be requested to label some formerly unlabelled data. In semi-supervised constraint clustering, the user may be requested to define some constraints, using his or her human expertise, after viewing the unlabelled data.

Semi-supervised learning with user-interaction is particularly useful in situations where there is a wealth of unlabelled data, but having a human set labels on the data is time consuming, and hence, expensive. LIDAR data definitely fits this situation, and hence, is ideal as a domain in which to leverage semi-supervised learning with user-interaction.

Instead of pre-producing and over-producing the expensive labelled training data to ensure that there is enough for training a supervised machine learning model, in semi-supervised learning with user-interaction, the training data can be supplemented and adjusted as needed.

Active Learning

If this requested human input is done in an iterative way, and, in each iteration, the human user is guided to provide the information that will likely be the most informative or clarify an area of the highest level of error, then this is defined as Active Learning. [BDW08]

This specific type of semi-supervised learning with guided user-interaction serves to focus the human’s user expertise on adding information that will be the most useful to the machine learning algorithm, with the hope that this will allow less supervised information to need to be acquired, because each piece of the supervised information is helpful, not redundant.

There are many schemes and algorithms for both semi-supervised machine learning and active learning, both for the underlying machine learning model and the way of interacting with and directing the user.
This thesis work uses active learning in two places — one for gathering constraints to help with clustering for segmentation to find potential classes of objects for labelling and one for selecting additional helpful training patches after viewing the SVM classification result on the whole LIDAR point cloud.
Chapter 3

Related Work

There have been many papers on the subject of the classification of airborne LIDAR data points. Many of them have focussed on the extraction of the “bare-Earth” for the DTM’s, as outlined in the survey by Meng, Currit, and Zhao [MCZ10]. There have been many other papers focussing on the identification and extraction of buildings and roads, such as the work by Sampath and Shan [SS10] and Boyko and Funkhouser [BF11]. Machine learning techniques, such as SVM’s, expectation-maximization, ANN’s, and committees have been applied to the classification of ground, buildings, roads, and forest canopies [Lod+06; LFH07; LX11]. There has been much less focus on non-ground, non-building, non-road, and non-forest classifications. This thesis work focusses on segmenting and classifying human-made objects separately from vegetation in urban environments, after the “bare-Earth” and buildings have been segmented, classified, and removed from the 3D airborne LIDAR point cloud.

3.1 Segmentation and Classification of Human-Made Objects in Urban Airborne LIDAR Data

Golovinskiy, Kim, and Funkhouser [GKF09] did focus on the classification of small urban objects, such as cars, street lights, and mail boxes, using shape descriptors and various machine learning algorithms (K-Nearest Neighbours, Random Forests, SVM’s). They presented various alternative designs and evaluated them. Feature vectors were contiguous-neighbour-based (number of points, volume, average and standard deviation...
CHAPTER 3. RELATED WORK

of height, and a spin image descriptor) and context-based (distance and orientation to the nearest street and a prediction grid of, for example, a car being near another car). They successfully recognized 65% of objects in the test area. Their approach was similar to the one being used in this thesis, except for the following aspects:

- Their input data was a LIDAR point cloud that was from a combination of ground-based mobile vehicle LIDAR scanners and airborne LIDAR scanners. This thesis only uses data from airborne LIDAR scanners.
- Different clustering/segmentation algorithms are used in this thesis.
- Different attributes are used in this thesis in the feature vectors (resultant from the different input source data — ground/airborne combination versus airborne-only). They used shape descriptors that were dependent on having a horizontal-view of the LIDAR point cloud.
- They leveraged the contextual information from a digital map. This thesis work restricts input information to only that which is intrinsic in the LIDAR point cloud.
- No semi-supervised or active learning was leveraged in this paper. This thesis work attempts to improve the accuracy of classifications with semi-supervised, active learning.

The details of the techniques in this thesis work are outlined in more detail in Chapter 4.

Chehata, Guo, and Mallet [CGM09b] achieved over 95% accuracy using Random Forests (Bagging with randomly-selected features to make Decision Trees) on full waveform, airborne LIDAR data with a density of 2.5 pulses per metre squared. They classified buildings, vegetation, natural ground, and artificial ground in an urban area. Chehata, Guo, and Mallet [CGM09a] used Random Forests for feature selection for urban classification. They noted that their classification for “artificial ground” included cars and traffic lights, classified together in the same class with the roads. No semi-supervised or active learning was leveraged in this paper, and it used a different machine learning model and different feature vectors than this thesis uses.

Chen and Zakhor [CZ09] segmented and classified only buildings versus trees in an urban LIDAR point cloud, with the intention of removing the tree data points before building polygonization/modelling for reconstruction. They achieved over 95% accuracy using a
rasterized, 0.2 metre resolution, two-and-a-half-dimensional (2.5D) version of their point cloud; edge-detection and region-growing in the segmentation; and Random Forests for classification. Their feature vector for the classification included the contiguous-neighbour-based attributes of the average standard deviation of the average height of returns in a pixel, the average edge density, and contour non-linearity (the count of the “spikes” in the second derivative of the following function: moving along the segments’ contour, of each outer pixel’s Euclidean distance to the centroid of the segment). No semi-supervised or active learning was leveraged in this paper, and it used a different machine learning model and different feature vectors than this thesis uses.

Höfle and Hollaus [HH10] and Höfle, Hollaus, and Hagenauer [HHH12] used full waveform LIDAR data to improve extraction of single trees in an urban LIDAR point cloud. They achieved over 95% accuracy using a rasterized, 0.5 metre resolution, 2.5D version of their point cloud; edge-detection with rules on height and number of returns in the segmentation; and several classification techniques. The earlier work [HH10] used a rule-based classification. The more recent work compared an ANN and a Decision Tree, with the ANN’s outperforming the Decision Tree. They tested various features in the classification that included contiguous-neighbour-based attributes based on full waveform information and on segment geometry: segment height mean and standard deviation; a ratio formed by the number of first, last, intermediate, and single returns; echo width mean and standard deviation; amplitude mean and standard deviation; backscattering coefficients means and standard deviations. The ratio formed by the number of first, last, intermediate, and single returns was the most effective discriminator. No semi-supervised or active learning was leveraged in this paper, and it used a different machine learning model and different feature vectors than this thesis uses.

Kim and Medioni [KM11] focussed a large part of their paper on the segmentation and classification of the ground and buildings in an urban LIDAR point cloud from a combination of airborne and ground-based scanning (though they note that the ground-based scanning points could be omitted). However, they also considered segmentation of small urban objects, such as cars and street signs. Using the points that remained unclassified after the ground, the roofs, and the vertical walls were classified, they first segmented linear structures, such as power lines and street lights. To do this, they voxel-ized the point cloud. In each voxel, they then calculated a tensor from the covariance of the points that resided in that voxel. From this, they determined a strength of linearity of the points in that voxel and
the orientation of the linearity. Voxels with a consistent, contiguous linear structure passing through them were then joined. They then segmented other objects, like antennas, satellite dishes, and chimneys on roofs. To do this, they again used voxels to create a bounding box of the height of the highest points in the area against the background of the ground, a roof, or a wall. This utilization of the context of the small object in relation to the large objects that they have already classified (building roofs, walls, and ground) and the use of tensors calculated from the covariance of the near-neighbourhood’s points for linearity detection could be tried for improvement of the feature vector in this thesis in future work (Section 6.1). This paper used different segmentation algorithms and attributes for urban objects than is used in this thesis. Classification of human-made urban objects was not considered in this paper — only segmentation. No semi-supervised or active learning was leveraged in this paper.

Roy and Maheux [RM11] presented a pipeline for fast automatic target detection and recognition of vehicles in an urban LIDAR point cloud. Their pipeline identified and removed the bare-Earth (assumed to be a plane), and then, projected the remaining points to the 2D ground plane and clustered the points in the 2D plane to determine the regions of interest, eliminating regions smaller or larger than certain thresholds. The pipeline then used the bounding box regions of interest to gather the 3D points in the volume above it into a volume of interest. The volumes of interest were classified into one of three vehicle types, using rules based on the volume of interest’s horizontal footprint’s length and width and the Euclidean distance to the volume’s points’ centroid. Then, a probability of correct recognition was calculated by comparing a quantized height map of the potential objects with a library of known vehicles. Though this system detected vehicles in urban LIDAR point clouds, it used very different techniques than is utilized in this thesis, including no semi-supervised or active learning.

Yao, Hinz, and Stilla [YHS11] achieved about an 85% accuracy in their vehicle versus non-vehicle classification of an urban LIDAR point cloud as a precursor to doing vehicle motion analysis. They employed adaptive mean-shift clustering for segmentation, using a feature vector of only the geometric x, y, and z-co-ordinates, and then, classified the segments with a binary SVM, using a feature vector for each segment that included the segment’s area, “elongatedness”, planarity, height at the centroid, and the height range. They then went on to detect the motion of a vehicle by taking the convex hull of the 2D horizontal projection of the vehicle, regularizing it, and then, analysing the distortion of the vehicle
CHAPTER 3. RELATED WORK

from a rectangular shape to a parallelogram because of the motion and the scanning angle. No semi-supervised or active learning was leveraged in this paper, and it used different feature vectors than this thesis uses for both the segmentation and classification.

3.2 Supervised Machine Learning with SVM for Classification of Airborne LIDAR Data

SVM’s have been used for many remote sensing applications on hyperspectral data, multispectral data, and LIDAR data. A sampling of papers that have successfully used SVM’s for LIDAR data classification are as follows:

- Lodha, Kreps, Helmbold, and Fitzpatrick [Lod+06] achieved over 93% accuracy using SVM’s with a RBF kernel (which was the most accurate in their test comparing various other kernel options) on LIDAR and image data with a 0.5 metre resolution to classify buildings, trees, and roads/grass on a university campus.

- Secord and Zakhor [SZ07] used SVM’s with a RBF kernel on airborne LIDAR and aerial images with a density of about four points per metre squared to classify trees/not trees in suburban environments in order to remove the trees before modelling the roofs. Feature vectors were segmented, contiguous-neighbour-based.

- Li, Gu, Han, and Yang [Li+07] used SVM’s with a RBF kernel on airborne LIDAR and aerial images with a 0.4 metre resolution to classify land cover (roads, buildings, shadows of buildings, trees, shadows of trees, grass, and bare land). Feature vectors were segmented, contiguous-neighbour-based.

- Alonso and Malpica [AM08] used SVM’s with a RBF kernel on airborne LIDAR and multispectral satellite imagery with a 2.5 metre resolution to classify low vegetation, high vegetation, natural ground, artificial ground, and buildings. The paper showed that including LIDAR data in addition to multispectral data significantly increased the classification accuracy.

- Dalponte, Bruzzone, and Gianelle [DBG08] used SVM’s with a RBF kernel on airborne LIDAR with a density of more than five points per metre squared and hyperspectral images with a 1 metre resolution to classify tree species in a forest. The authors
of the paper showed that, for this application, SVM’s had much higher classification accuracy than a K-Nearest Neighbour or a maximum-likelihood classifier. It performed well even when there were a high number of features and a low number of training samples. The authors of the paper also showed that the first LIDAR return showing vegetation height information was key to the quality of the classification.

- Koetz, Morsdorf, Van der Linden, Curt, and Allgöwer [Koe+08] used SVM’s with a RBF kernel on airborne LIDAR and hyperspectral images with a 0.1 metre resolution for land-cover and vegetation species classification at urban/forest interfaces for forest fire risk assessment and planning.

- Mallet, Soergel, and Bretar [MSB08] used SVM’s with a RBF kernel on full waveform, airborne LIDAR data with a density of 5 pulses per metre squared and another with a density of 2.5 pulses per metre squared. The paper classified buildings, vegetation, natural ground, and artificial ground in urban areas. Feature vectors were a combination of near-neighbour-based (difference of altitudes in a neighbourhood, residuals of plane fitting, deviation of the normal vector from a vertical direction) and waveform-based (difference of altitudes between first and last pulse, number of echoes, pulse amplitude, pulse width, and a pulse shape parameter), derived from the waveform processing. They achieved a 92% accuracy in a dense, urban area. The next year, however, they improved on their own result (95% accuracy) by using Random Forests (Bagging with randomly-selected features to make Decision Trees) [CGM09b].

- Frank, Pan, Raber, and Lenart [Fra+10] achieved over a 92% accuracy using a SVM with a RBF kernel on hyperspectral data, optical images, and airborne LIDAR data with a density of about 14 points per metre squared to monitor types of vegetation that had the potential of encroaching on power line corridors. Feature vectors were a combination of point/pixel-based and near-neighbour-based.

- Samadzadegan, Bigdeli, and Ramzi [SBR10] achieved over a 91% accuracy using multiple SVM’s with Bagging and using a committee vote for the final classification decision. Each SVM was trained with a randomly-selected subset of features. Feature vectors were a combination of point-based and near-neighbour-based. The airborne LIDAR was classified into buildings, trees, and ground in urban environments. They
also compared the one-against-all algorithm for multi-class SVM’s against the one-against-one algorithm and found that one-against-all was very slightly more accurate than one-against-one, but that one-against-one took much less time to train.

- Trinder and Salah [TS11] compared SVM kernels to find the one that worked best on airborne LIDAR and aerial images and compared SVM’s to statistical and neural classifiers, on four data sets (urban, rural, suburban/industrial, historic urban/industrial) with resolutions ranging from 0.09 to 0.5 metres in classifying buildings, trees, roads, and ground. Feature vectors were a combination of point/pixel-based and near-neighbour-based. They found that the RBF kernel slightly outperformed the linear, sigmoid, and polynomial kernels. They found that SVM’s slightly outperformed the statistical and neural classifiers tested.

- Zhao, Popescu, Meng, Pang, and Agca [Zha+11] used a SVM on airborne LIDAR with a density of 2.6 points per metre squared to classify types of forest vegetation. Feature vectors were a combination of point-based and near-neighbour-based. The paper showed that a SVM had a higher classification accuracy than a maximum-likelihood classifier.

- Zhang and Liu [ZL12] achieved over a 92% accuracy using a SVM on airborne LIDAR with a density of 4 points per metre squared to classify Australian tree species. Feature vectors were near-neighbour-based (maximum, mean, and standard deviation of heights, density of returns, mean and standard deviation of intensity of first and all returns in a near-neighbourhood of the region that is likely a tree crown).

- Mountrakis, Im, and Ogole [MIO11] provided a survey of SVM use for the classification of remote sensing data.

Note that these research studies required fully-labelled training data and had no user interaction to provide additional training data as-needed, unlike the semi-supervised, active learning approach detailed in this thesis. Within the semi-supervised, active learning framework for classification, this thesis work uses a SVM with a RBF kernel for the supervised machine learning model. It is selected as the machine learning model to be used within the active learning framework because of its success in supervised machine learning airborne LIDAR research, as demonstrated in these papers.
3.3 Unsupervised Machine Learning with Hierarchical Clustering for Segmentation of Airborne LIDAR Data

Hierarchical clustering has been used for many remote sensing applications on hyperspectral data, multispectral data, and LIDAR data. A sample of papers that have successfully used hierarchical clustering for LIDAR data segmentation are presented below.

To map vegetation in a flood-plane, Verrelst, Geerling, Sykora, and Clevers [Ver+09] used both LIDAR and multispectral data. They used agglomerative hierarchical clustering with average-linkage to create a hierarchy of vegetation classes. The feature vector included attributes from the multispectral data and the LIDAR data, including the minimum, maximum, mean, median, range, and standard deviation of z-values.

To segment individual tree crowns in a LIDAR point cloud, Lee, Slatton, Roth, and Cropper [Lee+10] used seed points at high elevation points that were likely the tops of trees and then, performed region-growing. The result may have been over-segmented, so then, they used agglomerative hierarchical clustering with centroid-linkage to merge segments until a distance threshold was reached. The distance measures were calculated as the Euclidean distance between the feature vector from each cluster’s centroid, the feature vector consisting only of the standard deviation of z-values in that cluster.

Note that the segmentation in these papers was unsupervised, i.e. there was no labelled training data, and also, it had no user interaction to provide any supervised guidance as-needed, unlike the semi-supervised, active learning approach detailed in this thesis. Within the semi-supervised, active learning framework for segmentation, this thesis work uses agglomerative hierarchical clustering with a Euclidean distance metric and complete-linkage for segmenting an unlabelled area of the LIDAR point cloud in order to find potential classes of objects for labelling.

3.4 Semi-Supervised Active Learning for Segmentation with Constrained Hierarchical Clustering

Klein, Kamvar, and Manning [KKM02] first proposed the use of semi-supervised learning with agglomerative hierarchical clustering, using must-link and cannot-link constraints. (Previous approaches to constrained clustering had used the k-means algorithm.) Their hierarchical clustering used complete-linkage (that is, in calculating the distance between
two clusters, the distance between each pairwise combination of data points between the two
clusters was considered and the maximum distance value was used to represent the distance
between the clusters). The distance between data points with a must-link constraint was
set to zero and the distance between data points with a cannot-link constraint was set to
be larger than the maximum distance value. The must-link constraints were propagated
to nearby points via a run of the shortest-paths algorithm. The cannot-link constraints
did not need to be propagated explicitly because they were implicitly propagated with
the complete-linkage. Their approach was tested on synthetic data sets and real-world
databases to segment different soy-bean diseases, irises species, and crab species, i.e. no
remote sensing data. As the number of constraints increased, the constrained hierarchical
clustering significantly outperformed the accuracy of constrained k-means clustering.

Klein, Kamvar, and Manning [KKM02] also proposed using an active learning approach
to gather the constraints for their semi-supervised learning with agglomerative hierarchical
clustering. They compared the random selection of two points on which to set either a must-
link or cannot-link constraint against their active learning approach in which a constraint
decision was requested of the user when a distance between two clusters exceeded a threshold.
Because the constraints were propagated, the distance may not have been exceeded again
for multiple cluster merges later. Tested by segmenting the databases of different soy-bean
diseases, irises species, and crab species, the active learning approach to gathering must-
link and cannot-link constraints dramatically decreased the number of constraints that were
needed to achieve a high level of accuracy.

3.4.1 Of Other Remote Sensing Data

Tuia, Kanevski, Muñoz-Marí, and Camps-Valls [Tui+10], Muñoz-Marí, Tuia, and Camps-
Valls [MTC12], and Tuia, Muñoz-Marí, and Camps-Valls [TMC12] first performed a hier-
archical clustering on the data points. Then, active learning was used, not to impose
constraints, but to select data points for labelling and then to use these labels to determine
the clustering granularity, that is, the correct level at which to partition the hierarchical
clustering tree. If the user labelled the points in a single cluster and this resulted in a
mixed set of labels in that cluster, then the cut-level for that branch was moved down to
where that cluster was split into two, as long as those sub-clusters would then have more
homogeneous labels. They tried various heuristics to use for selecting in which cluster to
select a random data point sample for labelling — sampling based on the size of the clusters,
sampling based on the size and uncertainty of the clusters, and a sampling based on only
the uncertainty of the clusters. Tuia, Kanevski, Muñoz-Marí, and Camps-Valls [Tui+10]
used spectral-only attributes in the feature vectors, but Muñoz-Marí, Tuia, and Camps-
Valls [MTC12] and Tuia, Muñoz-Marí, and Camps-Valls [TMC12] added spatial attributes,
improving classification accuracy. They successfully tested their technique for segmenting
and labelling hyperspectral and multispectral data of suburban and agricultural areas into
land-cover classes, and the active learning for sampling for labelling increased the accuracy
of the segmentation over random selection of samples for labelling.

Tuia, Muñoz-Marí, and Camps-Valls [TMC11] modified the technique by Tuia, Kanevski,
Muñoz-Marí, and Camps-Valls [Tui+10] to improve performance on large data sets of
hyperspectral data. The tree hierarchy was instead built by a different clustering scheme,
the bisecting k-means. The technique needed some labels to start, so it was not truly a
segmentation solution — more of a classification solution. The tree hierarchy was built up
with each bisection, looking at the confidence in the labels in a cluster and deciding whether
to bisect the clusters further. In this way, the bisection could have been stopped before the
full tree was built, when adequate confidence in the clusters was reached (and implying the
number of clusters to use). This improved the efficiency. Also, after the active learning
selection of a cluster, instead of selecting a random data point sample for labelling in that
cluster, the confidence in sub-clusters was used to select a data point within the selected
cluster for labelling.

3.4.2 Of Airborne LIDAR Data

No existing papers were found that focussed on semi-supervised, active learning for
segmentation of LIDAR data only. This thesis work uses semi-supervised learning for
segmentation with constrained hierarchical clustering on LIDAR data using slightly-soft
must-link and cannot-link constraints. This thesis work uses active learning to guide the
user to add constraints to clarify the segments with the most uncertainty. New constraints
between data points are added in region-based batches for efficiency of constraint addition,
leveraging the implicit spatial contextual information in remote sensing data.
CHAPTER 3. RELATED WORK

3.5 Semi-Supervised Active Learning for Classification

3.5.1 Of Other Remote Sensing Data

In 2008, Tuia et al. [Tui+08] noted that active learning had been applied rarely in the remote sensing domain. This paper used semi-supervised and active learning for classification to distinguish eight object classes (residential buildings, commercial buildings, asphalt, short vegetation, trees, soil, water, and drainage channels) in multispectral remote sensing images with a resolution of 0.6 metres.

First, a one-against-all multi-class SVM with a RBF kernel was trained using an initial random subset of the already-labelled data. Then, more multi-class SVM’s were trained using different initial random subsets of the already-labelled data. This technique, called Bagging, formed a committee of trained SVM’s. They were applied to the unlabelled data. The unlabelled data points that had the most disagreement in the committee of SVM’s, that is, the ones with the maximum entropy, were the ones that were presented to the user for labelling. The selected samples were labelled by a user, and then, the SVM’s were re-trained and re-run for the next iteration.

They compared their approach for active learning selection against margin sampling, which selected the points that were within the margins of the support vectors of a SVM for labelling. They also compared their approach against a random selection of additional samples for labelling. The various approaches all achieved a nearly 90% accuracy; however, their approach achieved the accuracy with less than half of the data samples required in the other two approaches.

In 2009, Bruzzone and Persello [BP09] also noted that active learning had only been marginally studied in the remote sensing domain. This paper used semi-supervised and active learning for the classification to distinguish buildings from other above-ground objects in multispectral remote sensing images with a resolution of 0.61 metres. First, a binary SVM with a Gaussian kernel was trained using an initial set of already-labelled data. Then, a batch of unlabelled points was automatically selected according to its uncertainty (related to the distance from the SVM hyperplane and comparison results with a validation set of data) and then, it was selectively reduced to achieve a good diversity. These selected, unlabelled points were then presented to the user for labelling. The selected samples were labelled by a user, and then, the SVM was re-trained and re-run for an improved prediction.
Since these papers, work on active learning has exploded in the remote sensing community and many, many new papers have been published.

Tuia, Volpi, Copa, Kanevski, and Muñoz-Marí [Tui+11] and Crawford, Tuia, and Yang [CTY13] provided an excellent survey of the active learning algorithms in use for the classification of remote sensing data.

Stumpf, Lachiche, Malet, Kerle, and Puissant [Stu+12] showed that for remote sensing images with implicit spatial contextual information, having the active learning query the user with small geographic regions, instead of single points, was more efficient, allowing the user to infer context and label sets of points according to that context, instead of batches of geographically disparate single points. This efficiency considered the human oracle’s time, in addition to the number of query iterations in the active learning. To reduce overall labelling time/costs, this thesis work employs the same region-based, active query technique, allowing users to label sets of points in small “training patches” as part of the active learning process. (Stumpf, Lachiche, Malet, Kerle, and Puissant [Stu+12] had a different supervised machine learning model for the classification — a Random Forest — and a different active learning selection metric — committee vote entropy for uncertainty and the standard deviation of the entropy for diversity — than is used in this thesis work.)

3.5.2 Of Airborne LIDAR Data

Wuttke, Schilling, and Middelmann [WSM12] presented a clearly-laid-out framework system design for active learning for classification — outlining that one module of the system was for selection of unlabelled points for labelling, another module was the supervised machine learning model that was iteratively re-trained with the additional data, another module was the oracle (typically a human user with some reference data) that labelled the selected data points, and another module did the feature extraction from the source data. In their implementation and testing of the framework, they tried to distinguish six object classes (low, medium, and high vegetation; water; and low and high non-vegetation) in hyperspectral and airborne LIDAR remote sensing data with a resolution of 0.8 metres. There was a time difference between the hyperspectral and LIDAR data collection, so moveable objects like cars were not considered in the classification. The attributes that they chose for their feature vectors were the Normalized Difference Vegetation Index and overall radiance from the hyperspectral data, and the height above ground and standard deviation of height in a near neighbourhood from the LIDAR data.
First, a K-Nearest Neighbour, supervised classification was run using an initial set of already-labelled data. Then, the certainty for each point was displayed to the user with a colour-coding. This allowed the user to select points for labelling that had the most uncertainty. The selected samples were labelled by a user, and then, the K-Nearest Neighbour, supervised classification was re-run with the expanded set of training data.

This approach was very similar to the one being used in this thesis for classification, except that

- instead of a K-Nearest Neighbour model for supervised classification, this thesis work uses a SVM.

- In both sets of work, the uncertainty of each point’s classification is displayed to the user to guide the user’s selection of uncertain, unlabelled points for labelling. Wuttke, Schilling, and Middelmann [WSM12] calculated the uncertainty as the distance to the nearest neighbour, whereas in this thesis work, the uncertainty is calculated from the multi-class SVM results.

- Wuttke, Schilling, and Middelmann [WSM12] had different attributes in the feature vectors — two attributes from the LIDAR data and two attributes from the hyperspectral data. This thesis work focusses on attributes from the LIDAR data (without the help of hyperspectral data) for the classification, and uses an optical remote sensing image only for the human user/oracle to help as ground truth data. The details of the techniques in this thesis work are outlined in more detail in Chapter 4.

No existing papers were found that focussed on semi-supervised, active learning for the classification of LIDAR data only.
Chapter 4

Active Learning for Semantic Labelling
of Human-Made Objects and Vegetation in an Urban Airborne LIDAR Point Cloud

Creating training data for classification of airborne LIDAR data is normally an expensive and time-consuming process. To accurately label the training data, either ground truth data must be gathered via field surveys, or, if high-resolution aerial photographs are available, a human user could use them for photointerpretation. In order to avoid getting poor end results, due to too little training data, organizations often label more training data than is actually needed — at a large expense of time and money.

Using semi-supervised, active learning in both the segmentation and the classification allows a minimal training data set to be created, tested, and then expanded as-needed in an interactive, iterative process. Through the active learning, users are guided to add training data in key areas, which reduces tedious labelling of training data that is redundant or less helpful.

In this work, the approach for the semantic labelling of human-made objects and vegetation in urban, airborne LIDAR point clouds uses active learning in two places — for both segmentation and classification.

In the segmentation, it utilizes active learning for gathering constraints to apply on the clustering to find potential classes of objects for labelling. Agglomerative hierarchical
clustering with should-link and should-not-link constraints segments a geographic area of the LIDAR point cloud, iteratively improving with user input. In this segmentation by clustering, the ideal number of clusters is automatically estimated by utilizing a modified version of the L-method.

The segmented and labelled points can then become training data and be used to train a supervised machine learning model (a SVM) to use for classification on the whole LIDAR point cloud. If the result of the classification on the whole LIDAR point cloud is not yet of the quality desired, the semi-supervised, active learning approach allows the user to iteratively add additional key training patches (or remove or refine others).

Using semi-supervised, active learning iterations for both segmentation and classification makes an efficient use (and re-use) of the human user’s expertise by guiding the user to add knowledge where it would be most helpful and then, re-using that knowledge over a larger geographical area.

4.1 Semi-Supervised, Active Learning for Segmentation and Labelling To Create Training Data

Starting with a visualization of the full geographic area of interest (such as a display of the first returns of the LIDAR point cloud, the last returns of the LIDAR point cloud, the intensity values of the first or last return, or, if available, a corresponding high-resolution aerial photograph of the area), the user makes a first guess at a good geographical area in which to develop training data. Figure 4.1 shows a user selecting a training patch with a varied mix of human-made objects and vegetation after observing the first and last returns and the intensity of the LIDAR point cloud, a corresponding aerial photo of the area, and the existing building and “vegetation and other objects” classifications.

The algorithm described in this thesis then calculates attribute values for a feature vector to represent each data point in this batch of points in the selected geographic area and automatically segments them using hierarchical clustering with the L-method for estimating the ideal number of segments.
Figure 4.1: Selecting the First Geographical Area in which to Develop Training Data Using a Visualization of the LIDAR Point Cloud's First Returns (a), Last Returns (b), Intensity of the Last Return (c), a Corresponding Aerial Photograph (d), the Points Classified as Buildings (in Red) (e), and the Points that may be Vegetation or other Objects (in Green) (f). Data is from Leading Edge Geomatics [Lea13] and analysis results are from Object Raku Technology’s Feature Type Interpreter [Obj13].
4.1.1 Hierarchical Clustering

The hierarchical clustering algorithm (as explained in Chapter 2’s section on Hierarchical Clustering) is ideal for this application for the following two reasons:

1. The hierarchical clustering algorithm is deterministic, which ensures consistent results in a single iteration of the algorithm. (There is no need to perform many clustering iterations to determine the stability of a particular potential solution, as would be needed if a non-deterministic clustering algorithm, such as k-means, was used.)

2. The ideal number of segments is not known in advance in this application. The hierarchical clustering algorithm allows the number of clusters/segments to be selected as one of the last steps of the clustering algorithm, after building a hierarchy according to distance measures between normalized feature vectors. The calculated distance measures can be used for estimating the best number of clusters. (There is no need to perform many clustering iterations to test different potential values for the number of clusters in order to try to determine the best number of clusters, as would be needed with clustering algorithms that require the number of clusters as an input, such as k-means.)

Early, experimental prototypes of this thesis work used k-means clustering; however, the fact that it was non-deterministic (hence, requiring many iterations to check for stability in a potential solution), and the fact that the number of clusters/segments was not known in advance in this application (hence, requiring many more iterations to test potential values in a search for the ideal number of clusters), necessitated so many iterations that the performance was prohibitive. Other clustering algorithms were investigated, and hierarchical clustering was found to be an ideal match of characteristics for this application.

Hierarchical clustering is much slower than a single iteration of k-means, but it requires only one iteration/execution in total, so it has a better total speed than the many iterations of k-means needed for this particular application. Holding the distance measures and the hierarchical tree that are created in the hierarchical clustering algorithm either requires a significant amount of memory or repeated re-calculation with a buffering system. In this thesis, all of the distance measures and the hierarchical tree are held in memory so that re-calculation and buffering are not needed. This does constrain the size of the data set that can be clustered at one time. The training patch’s small geographic area fits in memory.
CHAPTER 4. ACTIVE LEARNING FOR SEMANTIC LABELLING

[on a high-end personal computer (PC)] for the hierarchical clustering, making the use of the hierarchical clustering algorithm feasible for this application.

This thesis work uses a Euclidean distance measure and complete-linkage in the agglomerative hierarchical clustering.

4.1.2 Selecting the Ideal Number of Clusters with the L-Method

In agglomerative hierarchical clustering, distance measures are calculated for each data point with each other data point, and then clusters that have the closest distance measures are consecutively merged, with the clusters with the most similar (smallest) distance measure merged first, and clusters with the most dissimilar (largest) distance measure merged last. Hence, a full hierarchy of clusters and the distance measures associated with each cluster agglomeration have been created after an execution of the hierarchical clustering algorithm.

A graph can be created of the number of clusters versus the distance measure of the most-recently-combined pair of clusters at each possible number of clusters. This uses the already-calculated distance measures between the clusters that are merged at each level of the hierarchical tree. Figure 4.2 shows an example of the distance measures at each possible number of clusters from the training patch that is selected in Figure 4.1. (The values in the feature vectors are normalized before the hierarchical clustering, so the possible range of distance measures is between 0.0 and 1.0 here.)

At the point in the graph that clusters are being combined, even though there is a large distance measure between them, they likely should not be being combined anymore. The low-value, fairly flat area of the graph, where clusters are being combined with a very little distance measure between them likely are good merges of clusters, and all those merges should happen. The curved transition area between these two states is where the ideal number of clusters is located. This is sometimes referred to as the “knee” of the graph, as it is the area of highest curvature.

The L-method, proposed by Salvador and Chan [SC04], returns a suggested reasonable number of clusters selected in the “knee” area of the graph. The method uses two straight lines and searches for the best fit of those two lines to the data, using least-squares linear regression. The x-axis value at the point that those two best-fit lines intersect is the suggested number of clusters. Figure 4.3 shows the two lines in the positions of best fit in the graph of all of the possible number of clusters.
Figure 4.2: Graph of the Number of Clusters versus Distance Measures for Each Possible Number of Clusters for the Training Patch that was Selected in Figure 4.1
Figure 4.3: Lines in the Positions of Best Fit on the Graph of the Number of Clusters versus Distance Measures
Salvador and Chan [SC04] mention that some other techniques could be used to select the “knee” of the graph, such as at the point of the greatest magnitude or ratio change between two distance measures or a point with a large second derivative. These techniques, however, select the knee based on local features and could be prone to the effects of outliers. The L-method technique has a global scope.

The method proposed by Salvador and Chan [SC04] also uses iterative refinement to prevent a large number of data points in the low-distance-measure, flat area from skewing the results to the right. The iterative refinement first calculates a knee point using the L-method as described above, then reduces the size of the data set to be the calculated, suggested number of clusters, multiplied by two. It then re-calculates a knee point using the L-method again. It stops iterating in this manner when the suggested number of clusters does not decrease/move towards the left in an iteration. Figure 4.4 shows the improved lines of best fit with a reduced data set of consideration during the iterative refinement.

In their testing, the L-Method with iterative refinement outperformed existing methods (such as the Gap statistic [TWH01]) in both the accuracy of the number of clusters predicted and the speed of execution (as the Gap statistic and other methods must re-calculate distance measures for evaluation of each potential number of clusters).

This thesis implements a method similar to and inspired by this one by Salvador and Chan [SC04]. The number of clusters is determined by the following optimization:

\[
c_{\text{suggested}} = \arg\max_c RMSE_c
\]

\[
RMSE_c = \frac{c}{n} \times RMSE(L_c) + \frac{n-c}{n} \times RMSE(R_c)
\]

where \(RMSE\) is the root mean squared error; \(L_c\) is the set of the data points with the \(x\)-values for the number of clusters from 1...\(c\) and \(R_c\) is the set of data points with the \(x\)-values for the number of clusters from \(c+1\)...\(n\); \(n\) is the maximum number of clusters, equal to the number of data points in this application; and the range that \(c\) can be is \(c = 2...n - 2\) because the two lines must at least be comprised of two points each.

Figure 4.5 shows an initial segmentation of the selected training patch using hierarchical clustering and the L-method for the determination of the number of clusters. Without constraints and with the default attributes in the feature vector, some of the trees are erroneously segmented with the vehicles, and the trees appear to be over-segmented with their highly-varied heights.
Figure 4.4: Lines in the Positions of Best Fit with Iterative Refinement on the Graph of the Number of Clusters versus Distance Measures
CHAPTER 4. ACTIVE LEARNING FOR SEMANTIC LABELLING

Figure 4.5: Training Patch’s Initial Segmentation using Hierarchical Clustering and the L-Method (Overlaid on an Optical Image). LIDAR and optical data are from Leading Edge Geomatics [Lea13].

4.1.3 Active Learning to Gather Constraints

After the initial segmentation of the selected training patch with hierarchical clustering and the L-method, active learning can be used for gathering constraints to apply on the clustering to improve the segmentation of potential classes of objects for labelling. The user is guided to the particular clusters of the highest ambiguity, and then the user can specify should-link and should-not-link constraints, either specified as constraints on single points or as larger batches on clusters.

The user is directed to focus attention on the segments that are directly on either side of the decision boundary of the selected number of clusters. These are the segments that were either most recently combined to make that number of clusters or would be combined next in the agglomerative hierarchical clustering if there were to be one less number of clusters. Figure 4.6 highlights the clusters that are flagged to the user in the context of a hierarchical tree.
This active learning guidance to these clusters at the edge of the decision boundary focuses the user’s attention to the clusters that have the highest chances of needing adjustment.

Looking at the current segmentation in the geographical training patch and focussing on the particular clusters of the highest ambiguity, the user can then specify should-link and should-not-link constraints, either specified as constraints on single points or as larger batches on clusters.

The user can specify two or more points in the training patch that should have been placed in the same segment. These are point-level, should-link constraints. Alternatively, a user can specify two or more points in the training patch that should not have been placed in the same segment. These are point-level, should-not-link constraints.

Specifying point-level constraints can be time-consuming for a human user and may be especially frustrating if the segmentation appears obvious to the human user, but that user cannot specify it in any way except on a time-consuming, point-by-point basis. For
hyperspectral remote sensing data, Stumpf, Lachiche, Malet, Kerle, and Puissant [Stu+12] showed that batch labelling of points in one geographic region was more efficient than requesting individual points to be labelled or batches of points in disparate areas to be labelled and reasoned that humans can quickly view a geographic region and infer context to help with the labelling. As such, this thesis gives the human user the option of specifying constraints in larger batches, associated with one or more clusters. The user can specify that two or more whole clusters in the training patch should have been placed in the same segment. These are cluster-level, should-link constraints. Alternatively, a user can select one or more clusters in the training patch and then select a set of planes on which the selected clusters should have been split and placed in separate segments. These are cluster-level, should-not-link constraints. Allowing the user to specify constraints in controlled batches can reduce the number of active query iterations needed to achieve a satisfactory segmentation.

The agglomerative hierarchical clustering is then re-run with the constraints to re-segment the training patch area. This process can be repeated with the intent that the segmentation will improve in each iteration as a result of the semi-supervised help of the human user’s input.

4.1.4 Hierarchical Clustering with Constraints

This thesis work uses a semi-supervised version of agglomerative hierarchical clustering which takes into account constraints. It is very similar to the technique invented by Klein, Kamvar, and Manning [KKM02] for generic data sets, explained in the chapter on Related Work in the section on Semi-Supervised Active Learning for Segmentation with Constrained Hierarchical Clustering, but now it is applied to LIDAR data, enhanced with the option of batch-specification of constraints, and extended with the use of a modified L-method to select the number of clusters.

As is outlined in the paper by Klein, Kamvar, and Manning [KKM02] for generic data sets, after the distances are calculated between each data point with every other data point, the distances are adjusted based on the constraints. For data points that have a should-link constraint together, the distance between them is set to zero. For data points that have a should-not-link constraint between them, the distance between them is set to a very large value to represent infinity (but within computing numeric capabilities).
Batch-selection of constraints via cluster-level constraints in addition to point-level constraints for hierarchical clustering was first proposed by Davidson and Ravi [DR09] for generic data sets. This thesis uses a similar technique to the one used in their paper for the cluster-level, should-link constraints, but it deviates for the cluster-level, should-not-link constraints with the addition of a set of planes to define the splitting location, instead of just specifying the cluster-level, should-not-link constraints on existing clusters that have already been segmented into separate clusters. For clusters that have a should-link constraint between them, each pairwise point combination between the clusters has the distance between them set to zero. For clusters that have a should-not-link constraint with a set of 3D splitting planes, each pairwise point combination of all of the points in the involved clusters is checked to see if the two points lay on the same side or opposite sides of the set of planes. Because of the simplicity of the split being specified as simply a set of planes (and not a complex mesh), the criteria of a point being on the same or different sides of the set of 3D planes should be clarified. “In” is assumed to be the volume formed by the specified planes and the vertical and horizontal planes of the edges of the training patch that contains the origin of the x, y, and z-values. “Out” is the rest of the volume. If both points are in the volume, their distance measure is not changed. If one point is in the volume and the other is not, the distance between them is set to a very large value to represent infinity (but within computing numeric capabilities).

Similar to the technique employed by Klein, Kamvar, and Manning [KKM02] and Davidson and Ravi [DR09], using the transitivity and symmetry of should-link constraints, after the should-link constraints are set, the adjusted distance values are propagated to nearby points via a run of the Floyd-Warshall all-pairs-shortest-paths algorithm [Roy59; Flo62; War62]. The should-not-link constraints do not need to be explicitly propagated because they are propagated implicitly by the use of complete-linkage in the hierarchical clustering, where the largest distance value between each pairwise combination of data points between the two clusters is used to represent the distance measure between those two clusters. Hence, the very large value to represent infinity for should-not-link constraints are the ones that are propagated to represent a cluster.

The original and propagated constraint values facilitate the points that have a distance measure of zero to be clustered together and discourages points with a very large distance measure from being clustered together. All of the points that have a distance measure of zero between them will be the first ones to be combined in the agglomerative hierarchical
clustering algorithm, and the points that have a very large distance measure between them will be the very last to get combined.

Davidson and Ravi [DR09] note that there are cases where a set of must-link and cannot-link constraints contradict each other — e.g. a point that is part of a transitive closure of must-link constraints that also has a cannot-link constraint to one of the other points in that set. In this case, there is no valid clustering possible that respects all of these constraints. In this case, the algorithm by Davidson and Ravi [DR09] returns “no solution”. The algorithm by Davidson and Ravi [DR09] does not use active learning. Getting to a state of “no solution” is undesirable behaviour in an iterative, interactive, active learning environment. The method used in this thesis work where constraints are added in sequence in an active learning environment, with each iteration adding a new constraint and re-calculating the result, provides a way to avoid this situation. In a case where a new constraint contradicts a previous constraint, the new constraint takes precedence.

Davidson and Ravi [DR09] also discuss the situation of irreducibility — where at a certain level of agglomerations in the hierarchical clustering, no more merges are possible without violating a cannot-link constraint. At this point in their algorithm, moving beyond that to a smaller number of clusters is not possible — it is simply at an agglomeration dead end. The algorithm in this thesis mitigates and avoids this situation in two ways — one is by being more flexible with the constraints and the other is by avoiding the numbers of clusters in which this situation might occur.

This thesis uses a slightly softer version of the constraints than absolute must-link and cannot-link, similar to the technique used by Klein, Kamvar, and Manning [KKM02]. The should-link and should-not-link constraints can be violated in order to avoid an irreducible dead-end and create the full range of the number of clusters at all of the possible levels of the hierarchy; however, the constraint violations only occur at the top and bottom extremes of the hierarchy, and the distance measures on these agglomerations reflect this fact, with a zero-value distance measure on the agglomerations that are should-link and a very high penalty on the should-not-link agglomerations.

The technique used in this thesis work for selecting the ideal number of clusters, a modified L-method (defined in the next section), avoids selecting a number of clusters in these extreme levels of the hierarchy (that would violate these slightly-soft constraints). Extending the techniques of Klein, Kamvar, and Manning [KKM02] and Davidson and Ravi [DR09], which did not discuss automatic selection of the correct number of clusters, to
now select a particular, ideal number of clusters using a modified L-method, avoids both these zero distance measures and the very-large distance measures at these extreme lower and upper levels of the hierarchy where an irreducibility situation could occur with hard constraints.

### 4.1.5 Enhancement of the L-Method for Constraints

In agglomerative hierarchical clustering with constraints, distance measures are still calculated for each data point with each other data point, but then the distance measures are adjusted to reflect the should-link and should-not-link constraints. The resulting hierarchy of clusters and the distance measures associated with each cluster agglomeration after an execution of the hierarchical clustering algorithm have different values than they would have had without constraints. In particular, there are likely more zero-value distance measures and more “near-infinity” distance measures.

The graph of the number of clusters versus the distance measure of the most-recently-combined pair of clusters at each possible number of clusters illustrates this change. Figure 4.7 shows an example of the distance measures at each possible number of clusters from the training patch that was selected in Figure 4.1. This training patch has had some should-link and should-not link constraints added.

The values in the feature vectors were normalized before the hierarchical clustering, so the possible range of normal distance measures is between 0.0 and 1.0 here. Note that, on the left-hand-side, there are several points for numbers of clusters that have distance measures that are far greater than 1.0. These are the clusters that should not be linked and have a very high penalty applied on their joining. On the far right-hand-side, note that there are points for numbers of clusters that have distance measures of zero. These are the clusters that should be linked and so have distance measures of zero applied on their joining.

The curved transition area, the “knee” of the graph, is, again, where the ideal number of clusters is located. This avoids the areas of the graph where the number of clusters would cause a should-link or a should-not link constraint to be violated.

The existing L-method, as discussed in Section 4.1.2, already uses iterative refinement to prevent a large number of data points in the low-distance-measure, flat area from skewing the results to the right. As it reduces the set size from the right of the graph, the iterative refinement quickly eliminates the number of cluster values with distance measures of zero,
Figure 4.7: Graph of the Number of Clusters versus Distance Measures for Each Possible Number of Clusters after Hierarchical Clustering with Constraints for the Training Patch that was Selected in Figure 4.1, along with a Pop-Out Showing the Smallest Part of the y-Axis Expanded
all sitting at the far right of the graph, from consideration as a possible ideal number of clusters.

If the distance measures of "near-infinity" from the should-not-link constraints are included in the L-method optimization, then these artificially-very-high distance measures will typically skew the best-fit lines and move the estimation of the number of clusters to the left. Figure 4.8 shows an example of an artificially skewed graph.

![Skewed Lines in the Positions of Best Fit on the Graph of the Number of Clusters versus Distance Measures. The should-not-link constraints have artificially skewed the left best fit line to the left.](image)

To remedy this, the clusters that are forced to be separate as part of a should-not-link constraint are identified before the best-fit line optimization and are not included in the best-fit line optimization so that they will not skew the estimation. They are then added
back in after the ideal number of clusters is determined with the omission of the forced separations.

The number of clusters is, hence, determined by the following, slightly adjusted algorithm:

\[
c_{\text{suggested}} = snl + \arg\max_c RMSE_c
\]

\[
RMSE_c = \frac{c - snl}{n - snl} * RMSE(L_c) + \frac{n - c - snl}{n - snl} * RMSE(R_c)
\]

Where \(snl\) is the number of clusters that must exist because of the should-not-link constraints; \(RMSE\) is the root mean squared error; \(L_c\) is the set of the data points with the \(x\)-values for the number of clusters from \((snl + 1)...c\) and \(R_c\) is the set of data points with the \(x\)-values for the number of clusters from \(c+1...n\); \(n\) is the maximum number of clusters, equal to the number of data points in this application; and the range that \(c\) can be is \(c = snl + 2...n - 2\) because each of the two lines must at least be comprised of two points at the minimum.

Figure 4.9 shows the corrected graph, with the skewing removed.

Figure 4.10 shows a segmentation of the selected training patch that has had some should-link and should-not link constraints added. The trees are now segmented separately from the vehicles and the rounded shed on the left-hand edge, and the trees have been encouraged to be segmented together despite their highly-varied heights. Some fences are segmented in with the trees and the shed, though. More constraints could continue to be added to separate the fences.
Figure 4.9: Skew Corrected with the Enhancement of the L-Method for Constraints — Lines in the Positions of Best Fit on the Graph of the Number of Clusters versus Distance Measures
4.1.6 Neighbourhood-Based Attributes Assigned to a Point’s Feature Vector

The segmentation with hierarchical clustering, of course, requires each data point in the selected geographic area to have its pertinent characteristics represented as attributes in a feature vector. The right attributes must be used so that the distance measures calculated between the different feature vectors distinguish human-made objects and vegetation into the right segments.

The default set of attributes used in this thesis is a combination of point-based attributes, immediate-neighbour-based attributes, and contiguous-neighbour-based attributes. The addition of attributes that are based on context is left for future work, as listed in Section 6.1. The default set of attributes used in this thesis is the following:

- **Point-Based Attributes:**
  - Height above the z-value set as ground for the LIDAR signal’s last return
  - Difference in z-value between the first and last LIDAR signal return
• Immediate-Neighbour-Based Attributes:
  - Range of z-values of the LIDAR signal’s last return in the immediate neighbourhood

• Contiguous-Neighbour-Based Attributes:
  - Number of points in the contiguous neighbourhood
  - Range of z-values of the LIDAR signal’s first return in the contiguous neighbourhood
  - Normal vector of the LIDAR signal’s last return values in the contiguous neighbourhood
  - Average z-value of the first LIDAR signal return

These attributes have been selected through manual feature selection and empirically found to be effective for distinguishing human-made objects and vegetation. Many other attribute combinations are possible in the software developed for this thesis, as listed in the section on Template Creation Attributes and Weights in Appendix B. Further experimentation with attribute combinations and inclusion of automatic feature selection within the active learning for segmentation are left for future work, as listed in Section 6.1.

Using neighbourhood-based attributes helps reduce the effects of noise, as well as provide some local context on a point’s location. The immediate-neighbour-based attributes for a certain point are calculated using that point and all other points near to it, within a certain radius in the x-y direction. This radius is derived from the data and is set to be the average resolution from one point to the next. If the LIDAR data has been gridded to 2.5D, then this effectively includes the point and its eight neighbouring pixels’ data points, if there are above-ground, non-building points in those locations, as illustrated in Figure 4.11.

Using these immediate neighbours, attributes such as a point’s normal and planar error can be calculated, as well as some average, effectively “smoothed” values of the point-based attributes.

The contiguous-neighbour-based attributes for a certain point are calculated using that point and all other points that are deemed to be contiguous with it. Points are considered contiguous if they are within a certain radius, in the x-y direction, of any of the other points that have already been found to be contiguous, and also within a certain threshold, in the
Figure 4.11: Immediate Neighbours of the Starred Point
z-direction, of nearby contiguous points. The radius for the x-y direction is derived from the LIDAR data based on the average resolution from one point to the next. The z-direction threshold is set to 1.0 metres. It is set based on the premise that established non-ground objects are typically considered contiguous when they are within that granularity of height, e.g., the cab of a truck to the cargo area of a truck or the attached bench and table surface of a picnic table. If this premise does not hold for a certain data set, this value can be adjusted in the software’s graphical user interface (GUI).

An example of a contiguous neighbourhood is illustrated in Figure 4.12.

Finding the contiguous neighbourhoods of the points for the purpose of calculation of attributes for the feature vectors is really another segmentation, before the main segmentation. The contiguous neighbourhoods are quickly calculated using a single-linkage, hierarchical clustering with a Euclidean distance measure: First, Euclidean distances are calculated between every data point using only the x and y values of each data point. Then, the difference in z-value between each data point is considered. If the z-value difference exceeds the threshold, then the x-y Euclidean distance for those data points is set to be a very large value to represent infinity (but within computing numeric capabilities) so that those points will not be able to be a linkage point for linking those points in a contiguous neighbourhood. Then, hierarchical clustering is performed using these distance measures. Use of single-linkage ensures that the distance measures calculated between clusters and points are the smallest/nearest ones possible in evaluating whether there is a near distance between the clusters and points in the agglomeration. Looking at the distance measures of the hierarchical tree, the number of clusters is selected to be at the location where the x-y Euclidean distance measure crosses the radius threshold value. The resulting segmentation is the set of contiguous neighbourhoods that is used to calculate attribute values.

Using these contiguous neighbourhoods, attributes, such as a point’s contiguous neighbourhood’s number of points, dimensions, normal, ranges, and averages, can be calculated.

Note that, though the neighbourhood-based attributes are calculated using a set of points, they are assigned to and associated with the feature vectors that represent single data points. A single data point’s feature vector may contain some attributes that are point-based, some that are immediate-neighbour-based, and some that are contiguous-neighbour-based. This is different than segmenting the points into contiguous neighbourhoods and then just using one feature vector to represent each contiguous neighbourhood. Though there is a detriment to performance in regards to speed in having a feature vector associated with
Figure 4.12: Contiguous Neighbours of the Starred Point
each data point, it provides more flexibility in using combinations of scope for attributes and prevents "locking" a point into a segment before the main segmentation clustering. This allows some freedom for a point to move between possible contiguous segments if some other attributes are stronger. (If the initial segmentation defining the neighbourhoods was incorrect, a point could still "break-out" if it really did not fit, based on other attributes.) Hence, the first segmentation, just to find contiguous neighbours, is a "soft" segmentation — more of a guidance than a set partition.

The feature space of each feature vector is normalized so that the different offsets and ranges on the different attributes do not arbitrarily skew the importance of one attribute versus another.

The weight vector specifies weights for each type of attribute in the feature vectors in order to explicitly set certain attributes to have more or less relative importance in the distance measures calculation. The weights can be specified from 0.0 to 1.0, but the actual importance depends on the ratio of these weights to all of the attributes’ weights that have been specified. The weights for the default set of attributes used in this thesis are listed in Table 4.1.

These attribute weights, with the set of data points, each represented by a normalized feature vector, are then used for the segmentation with hierarchical clustering.
Table 4.1: Default Attributes and Weights

<table>
<thead>
<tr>
<th></th>
<th>Weight</th>
<th>Relative Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point-Based Attributes:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Height above the z-value set as ground for the LIDAR signal’s last return</td>
<td>1.0 of 4.85</td>
<td>0.21 of 1.0</td>
</tr>
<tr>
<td>Difference in z-value between the first and last LIDAR signal return</td>
<td>0.55 of 4.85</td>
<td>0.11 of 1.0</td>
</tr>
<tr>
<td><strong>Immediate-Neighbour-Based Attributes:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range of z-values of the LIDAR signal’s last return in the immediate neighbourhood</td>
<td>0.81 of 4.85</td>
<td>0.17 of 1.0</td>
</tr>
<tr>
<td><strong>Contiguous-Neighbour-Based Attributes:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of points in the contiguous neighbourhood</td>
<td>0.5 of 4.85</td>
<td>0.10 of 1.0</td>
</tr>
<tr>
<td>Range of z-values of the LIDAR signal’s first return in the contiguous neighbourhood</td>
<td>0.52 of 4.85</td>
<td>0.11 of 1.0</td>
</tr>
<tr>
<td>Normal vector of the LIDAR signal’s last return values in the contiguous neighbourhood</td>
<td>0.16 * 3 components = 0.48 of 4.85</td>
<td>0.10 of 1.0</td>
</tr>
<tr>
<td>Average z-value of the first LIDAR signal return</td>
<td>0.99 of 4.85</td>
<td>0.20 of 1.0</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td>4.85 of 4.85</td>
<td>1.0 of 1.0</td>
</tr>
</tbody>
</table>
4.1.7 Labelling

After enough constraints have been added to the training patch’s clustering to achieve a segmentation that is to the human user’s satisfaction, then the human user assigns each segment a label. If the training patch is over-segmented, multiple segments can just be assigned the same label. (Hence, over-segmentation is preferred to under-segmentation in this application.) After all segments are labelled, the training data in this training patch is then ready to be used to train a machine learning model for classification of the whole geographical area of the LIDAR point cloud. Figure 4.13 shows a segmented, labelled training patch.

Figure 4.13: Training Patch’s Segmentation Improved with Constraints and then Labelled (Overlaid on an Optical Image). LIDAR and optical data are from Leading Edge Geomatics [Lea13].

4.2 Semi-Supervised, Active Learning for Classification

After the active learning iterations are complete for segmentation of a training patch and the segments in the training patch have been labelled, the user can use these labelled data points to train a supervised machine learning model for classification of the rest of the geographic area.
The multi-class SVM supervised machine learning model (as explained in Chapter 2’s section on the Support Vector Machine) with a RBF kernel has been found to be a highly effective supervised classifier in other airborne LIDAR research, as listed in Chapter 2’s section on Supervised Machine Learning.

4.2.1 Training Data Feature Vectors

The input training data examples are the set of data points from the training patch, each with a feature vector of its full set of attributes and its object class name from the list of labelled segments. The full set of possible attributes is used in the data point’s feature vector, except for the data point’s x and y values, for which the explicit values are only locally relevant, and hence, are not constructive for a classification on a larger geographical area. This full set of attributes is used in the data point’s feature vector, not just the same attributes that were used for the segmentation, in case there is a pertinent, distinguishing attribute that was not included in the attributes used for segmentation, and the segmentation was only successful at distinguishing between types of objects because of the constraints added. Supplying values for all of the possible point-based, immediate-neighbour-based, and contiguous-neighbour-based attributes as inputs to the SVM for each training data point allows the SVM to select to use and to give a high-importance weight to any attribute that exhibits values that distinguish two classes in the training data set.

With sixty-five attributes and the number of data points depending on which training patch selected — with an order of magnitude from zero to 10,000 data points, the “curse of dimensionality”, also called the Hughes phenomenon [Hug68], could be a concern when the number of data points is small compared to the number of attributes. There could indeed be enough attributes that each attribute could be tuned in the machine learning model, one to match each data point exactly, overfitting the model to exactly match the small data set, and possibly causing a reduction in the generality of the model. In this thesis, this problem is mitigated in two ways. The first way is simply by noting that this case is rare and that it is easily corrected in this iterative, active learning framework. It is more common for a training patch to have a number of points in the higher end of the range. Executing a full classification with such a small number of training data points would be rare and may indeed produce inferior results; however, this can easily be corrected by active learning — adding another training patch and re-running the classification, as explained in more detail in the sections that follow. The second way that the “curse of dimensionality” is mitigated is
by using a SVM as the machine learning model. SVM’s have been found to remain effective even with a high-dimension of attributes with a low number of training data samples; it is only minimally affected by the Hughes phenomenon [MB04].

The immediate-neighbour-based and contiguous-neighbour-based attributes are calculated in the same way as explained in Section 4.1.6 for the attribute calculation for segmentation. For the same reasons as for the segmentation, the neighbourhood-based attributes, though they are calculated using a set of points, are assigned to and associated with the feature vectors that represent single data points — a single data point’s feature vector contains some attributes that are point-based, some that are immediate-neighbour-based, and some that are contiguous-neighbour-based. Having a feature vector associated with each data point, instead of only one feature vector for each segment, provides more flexibility in using combinations of scope for attributes and prevents the need to divide the whole geographic area of the LIDAR point cloud into segments before the SVM classification. Segmenting the points before the SVM classification could have the unwanted consequence of potentially “locking” a point into a segment according to a more limited set of attributes before utilizing the trained SVM with the full set of attributes. Instead, using an attribute vector for each data point separately, but with neighbourhood information within it, allows the contiguous neighbours to be only a “soft” segmentation/guidance for the SVM and lets the SVM make its best prediction for each point on the full information that it has.

Before the feature vectors for each data point are supplied to the SVM for training, the feature space of each attribute in the feature vectors is normalized so that the different offsets and ranges on the different attributes do not arbitrarily skew the importance of one attribute versus another.

### 4.2.2 Support Vector Machine Classification

The training data is then used to conduct a grid search of possible values for the two configurable parameters for the SVM — one, $\gamma$, that controls the shape of the RBF kernel and the other, $C$, that controls the strength of the penalty for being on the wrong side of the SVM soft-margin decision boundary — to empirically test and find the values for which the SVM achieves the highest accuracy on the training data set, using five-fold cross-validation for the training and testing/validation.

The best $\gamma$ and $C$ are then used with the full set of training data feature vectors for each data point and its associated label to train the multi-class SVM that will be used for classification of the full geographic area.
After training, the multi-class SVM has a formed model for prediction. Input feature vectors given to it will result in a prediction based on that model. As the model is formed and applied with an “even hand” to any new inputs, the rest of the large LIDAR point cloud geographic area can be fed to the SVM in tiles that fit in the memory of a high-end PC. The feature vectors that are the inputs for the prediction are calculated in the same way as the feature vectors were for the training of the SVM. Because those input feature vectors contain neighbourhood information, care must be taken to ensure that the data points near the periphery of the tiles are not deprived of their neighbourhood information that may lay in an adjacent tile. To prevent this, an overlap of 25% of the size of the tile, on all sides, is used just for the calculation of the feature vector attribute values. Then, the prediction in the SVM is executed just on the original tile size. Figure 4.14 shows a set of classification results after the multi-class SVM has executed predictions on all of the tiles in the geographic area. This SVM has been trained with only the one training patch from Figure 4.13 and clearly does not yet have enough training data to generalize the SVM model beyond the one test patch. More training data is needed to have accurate results. There are very few vehicles and fences identified. Most of the points are classified as vegetation — some correctly and some erroneously.

A standard SVM only predicts class labels — no probabilities of correctness associated with those labels. Chang and Lin [CL11] implement an extension to the standard multi-class one-against-one SVM, using the technique by Wu, Lin, and Weng [WLW04] to also output a probability for each data point’s predicted label. This thesis uses the implementation of Chang and Lin [CL11] to output a posterior probability associated with each predicted classification.

After classification of the full geographic area of the LIDAR point cloud, the probabilities of each data point’s classification are displayed to the user in order for the user to visualize which areas have a low confidence in the classification. Figure 4.15 shows the confidence of each data point from the same classification as Figure 4.14. The confidence is shown as a grey-scale value with lighter values being more confident and darker values being less confident.

Some of the areas where the vehicles were misclassified indeed have a low level of confidence. More training data is needed there.
Figure 4.14: Classification Results with Only One Training Patch — Vehicles are labelled pink, fences are labelled purple, sheds are labelled green, and vegetation is labelled gold (as also shown in the legend in Figure 5.7a). LIDAR and optical data are from Leading Edge Geomatics [Lea13].
Figure 4.15: Probability Visualization of Classification Results with Only One Training Patch. LIDAR and optical data are from Leading Edge Geomatics [Lea13].
4.2.3 Active Learning to Improve Training Data

After the initial classification of the full LIDAR point cloud’s geographic area with the SVM and the initial set of training data, active learning can be used to expand and improve the set of training data to try to improve the SVM classification. The user is guided to the data points of the highest uncertainty, and then the user can add a new training data patch in that location (or improve or delete an existing one there).

The user is directed to focus attention on the data points that are marked by the posterior-probability-outputting SVM as having a low probability on the given classification. Looking at the probability greyscale and the current classification results, the user can focus on the particular areas where the data points have the highest uncertainty and may have been misclassified.

Figure 4.16 highlights an area where vehicles and sheds have been misclassified, and also, those data points are marked as having a low probability with a dark greyscale value.

![Figure 4.16: Misclassification of Vehicles and Sheds in Areas Marked as Having a Low Confidence. Points that are classified as vehicles are labelled pink, fences are labelled purple, sheds are labelled green, and vegetation is labelled gold (as also shown in the legend in Figure 5.7a). LIDAR and optical data are from Leading Edge Geomatics [Lea13].](image)

This active learning guidance to these areas of highest uncertainty focusses the user’s attention to geographic areas that have the highest chances of being wrong and need clarification with more training data examples in the supervised machine learning model.
In the active learning iterations, specifying new sets of training data in small geographical batches, the training patches, instead of single points or a set of points in disparate areas is more efficient in total (human plus computing) time taken, as shown by Stumpf, Lachiche, Malet, Kerle, and Puissant [Stu+12] for hyperspectral data. Allowing the user to add training data in controlled geographic batches leverages the spatial context inherent in remote sensing data, enabling the user to use the context to determine the identity of the sets of points quickly. It can reduce the number of active query iterations needed to achieve a satisfactory classification.

The classification is then re-run with the updated set of training data. This process can be repeated, with the intent that the classification will improve in each active learning iteration with the semi-supervised help of the human user’s input. Figure 4.17 shows a set of classification results after five more training patches have been added and the classification has been re-run, resulting in improved results. Figure 4.18 shows a set of classification results after seven more training patches have been added and the classification has been re-run, again, achieving improved results. The classification can continue to be improved by adding more training patches.

Utilizing two levels of active learning — one set of active learning iterations in the segmentation and labelling of training data and another for the classification with a SVM — makes efficient use of a human user’s time and expertise in a semi-supervised approach to improving the semantic labelling of an airborne, urban LIDAR point cloud. After utilizing all of the labelled training data areas to train a SVM and performing a prediction over a larger geographic area, new training areas (with their own sub-iterations of active learning) can be added in key geographical areas exhibiting errors, and/or the previous areas can be removed or improved, and the SVM training and prediction can then be re-run.
Figure 4.17: Classification Results Improved With Only Five Iterations of Semi-Supervised, Active Learning, Adding One More Training Patch In Each Iteration — Passenger Vehicles (Pink), Buses (Bright Green), Sheds (Forest Green), Power Lines (Emerald Green), Fences (Pink-Purple), Lamp-Posts (Dark Purple), Garbage Bins (Light Purple), Not Interested (Turquoise), and Vegetation (Gold) (as also shown in the legend in Figure 5.7a). LIDAR and optical data are from Leading Edge Geomatics [Lea13].
Figure 4.18: Classification Results Improved With Only Seven Iterations of Semi-Supervised, Active Learning, Adding One More Training Patch In Each Iteration — Passenger Vehicles (Pink), Buses (Bright Green), Sheds (Forest Green), Power Lines (Emerald Green), Fences (Pink-Purple), Lamp-Posts (Dark Purple), Garbage Bins (Light Purple), Not Interested (Turquoise), and Vegetation (Gold) (as also shown in the legend in Figure 5.7a). LIDAR and optical data are from Leading Edge Geomatics [Lea13].
Chapter 5

Results

The techniques presented in this thesis work are evaluated here in several ways — the computing performance, the active learning for segmentation, the active learning for classification, and the region-based selection.

5.1 Test Data

For these results, the data that is used is a LIDAR point cloud from Leading Edge Geomatics [Lea13] that covers 0.53 kilometres squared (1 kilometre by 0.53 kilometres) of an urban area in New Brunswick, Canada, at about 45.84° North and 66.50° West. It has 4 170 093 points, with 62% of them being first returns and 17%, 13%, 6%, and 2% percent of them being second, third, fourth, and fifth returns, respectively. The z-values range from 16 to 610 metres. Figure 4.1 shows most of this geographic area. Because this thesis work focusses on distinguishing human-made objects from vegetation after the ground and buildings have been segmented, classified, and removed from the point cloud, the test data required by this thesis work starts where the ground, building, and “vegetation and other objects” analysis results from Object Raku Technology’s Feature Type Interpreter [Obj13] end. These existing analyses currently operate in a rasterized, 2.5D format, so that is the type of test data on which these results are based. It is rasterized into a 2.5D height map for both the first and last return, in addition to the intensity of the last return. The bare-Earth, building, and “vegetation and other objects” analysis results are labelled geoTIFF raster files. For this test data, they have been rasterized at a resolution to have 4 pixels by 4 pixels per metre squared, each pixel being 0.25 metres in length and width, for a total
extent of 4005 pixels by 2120 pixels. The points classified as “vegetation and other objects” that form the input test data for these results contain 1 800 319 first return and 1 800 319 last return populated points.

The training patches are 100 pixels by 100 pixels sub-areas, so are 25 metres by 25 metres. They could contain up to 20 000 points.

5.2 Software Developed

All of these results are generated using the software that has been developed to implement the techniques of this thesis, added on to Object Raku Technology’s Feature Type Interpreter [Obj13]. Details of the software requirements and design are included in Appendices A and B.

5.3 Performance

The computing performance of the software developed for this thesis work has been tested on a PC with an Intel i7-740QM Quad Core Mobile 1.73/2.93 GHz CPU with 8 GB DDR3 1333 MHz RAM, with 64-bit Windows 7.

The initial segmentation of a training patch takes approximately 13 seconds. Later iterations of the segmentation with constraints can take progressively longer, approximately up to two minutes, depending on the number of constraints and the number of points involved in the constraints. The number of point-level constraints increases the execution time approximately linearly. Cluster-level constraints, which may need to loop through the points in combination with each other, may increase the execution time quadratically in proportion to the number of points in the involved clusters. Propagation of the constraints, done with the Floyd-Warshall all-pairs-shortest-paths algorithm, is polynomial on the number of points and does not increase with the number of constraints. Figure 5.1 shows a plot of execution time versus the number of constraints for point-level constraints and cluster-level constraints.

Training the SVM machine learning model with one training patch and doing a grid search to find the best parameters for the SVM take approximately eight minutes. The execution time for training the SVM machine learning model with more training patches increases as the number of training data points from the training patches increases. Figure 5.2 shows a plot of this.
Figure 5.1: Segmentation Execution Time Versus Number of Constraints — Though the times are scattered, likely due to the processor’s turbo-boost capability, the trend lines show the general trend. This training patch had 2706 first and 2706 last return populated points.
Figure 5.2: SVM Parameter Selection and Training Execution Time Versus Number of Training Data Points — Though the times are scattered, likely due to the processor’s turbo-boost capability, the trend line shows the general trend.
The time to execute the classification does not depend on the number of training patches or training data points. It takes about one hour on this computer on this test data set.

5.4 Active Learning Guidance versus Random Selection for Segmentation

The accuracy of a segmentation is improved with the addition of constraints, as shown qualitatively in Figure 5.3 and quantitatively in Figure 5.4. Figure 5.3 shows several training patches. For each, it shows an optical image of the area for comparison and then shows the initial segmentation of the LIDAR data without any constraints. As constraints are added, the quality of the segmentation improves. Figure 5.4 also shows the improvement as constraints are added. It shows the improvement of the accuracy by adding constraints using the active learning guidance as implemented in this thesis work in comparison to the improvement of the accuracy by adding constraints at randomly-selected data points. For both of the plot lines, the human user is used to decide on whether to apply a should-link or a should-not-link constraint, but the difference in the two plot lines is the active learning guidance to advise/give additional information to the human user’s expertise. The plot lines for the active learning have a steeper increase to a higher level of accuracy than the random-selection plot lines. With the random selection of points for constraints, the accuracy still increases but not as quickly with the same number of constraints. (In the chart labelled “b”, the lines for active learning and random selection are directly on top of each other. Hence, for this one training patch, random selection was equal to active learning. However, this training patch only took one constraint to achieve an adequate segmentation, so the comparison set is too small to show the trend.) In both of these figures, only point-level constraints are used.

The truth data for assessing this accuracy was created with photointerpretation of the optical images. Over-segmentation was not penalized in the assessment of accuracy for segmentation. As long at the points could be labelled correctly, they were considered to be accurate for the segmentation, as, in this application, as long as the points can be labelled accurately, they will be equally effective in training the SVM. Of course, over-segmentation is not ideal due to the tediousness of labelling. The qualitative results show that the over-segmentation, where present (Training Patch 2 with 0 constraints and all of the Training Patch 3 results), is not extensive.
CHAPTER 5. RESULTS

Figure 5.3: Training Patches Improving With Active Learning as Only Point-Level Constraints are Added (Overlaid on an Optical Image). LIDAR and optical data are from Leading Edge Geomatics [Lea13].
CHAPTER 5. RESULTS

(a) Training Patch 1

(b) Training Patch 2
Figure 5.4: Chart of Training Patches’ Accuracy Improving as Only Point-Level Constraints are Added With Active Learning to Guide Selection of Points for Constraints (Diamond Data Points) Compared to Random Selection of Points for Constraints (Square Data Points)
5.5 Active Learning Guidance versus Random Selection for Classification

The accuracy of a classification is, as expected, improved with the addition of more training data patches, as shown qualitatively in Figure 5.5. Figure 5.5 shows several sub-areas of the full geographic area. For each, it shows an optical image of the area for comparison and then shows the initial classification of the LIDAR data with only one training patch. As training data is added, the quality of the classification improves. It shows the improvement of the accuracy by adding more training data patches using the active learning guidance as implemented in this thesis work.

For comparison, Figure 5.6 shows the same sub-areas that are a result of a classification with the same number of training patches as the best results in Figure 5.5. These training patches, however, were selected at random instead of using active learning. In both figures, the human user was used to assist with segmentation and labelling of the training patches, but the difference was the active learning guidance to advise/give additional information to the human user’s expertise for selection of a training patch. These results from the classification that used training patches that were selected with active learning are more accurate than the classification that randomly-placed training patches because the training patches that were selected with active learning are assured to be helpful ones to the machine learning model.

Due to the unavailability of ground truth data on the full geographic area and the fact that a classification accuracy increase resulting from the provision of more training data to the machine learning model is an obvious consequence, only qualitative examples (not quantitative) are provided for the active learning iteration improvement of the classification.
CHAPTER 5. RESULTS

Figure 5.5: Classification Results With Iterations of Active Learning Increasing the Amount of Training Data. The colour-coding is listed in Figure 5.7a. LIDAR and optical data are from Leading Edge Geomatics [Lea13].
CHAPTER 5. RESULTS

Figure 5.6: Classification Results With Active Learning Versus Randomly-Selected Training Patches. The colour-coding is listed in Figure 5.7. LIDAR and optical data are from Leading Edge Geomatics [Lea13].

Figure 5.7: Colours Used for the Classification Results to Label Types of Objects
5.6 Region-Based versus Point-Based Selection for Active Learning for Segmentation and Classification

Batch specification of constraints on a cohesive geographic area of airborne LIDAR data is more efficient for active learning for segmentation than specification of constraints on single points in disparate geographic areas. Utilizing the human’s ability to infer the spatial context in a geographical area and applying constraints on whole segments of points improve the accuracy of the segmentation in fewer iterations than by applying constraints on single points. This is shown quantitatively in Figure 5.8. It shows the improvement of the accuracy by adding both point-level and cluster-level constraints using the active learning guidance as implemented in this thesis work in comparison to the improvement of the accuracy by adding point-level constraints only. For both of the plot lines, the human user is used to decide on whether to apply a should-link or a should-not-link constraint using active learning guidance to advise/give additional information to the human user’s expertise, but the difference in the two plot lines is that in the one, only point-level constraints are allowed, and in the other, cluster-level constraints are allowed as well. The plot line that allows cluster-level constraints as well has a steeper increase to a higher level of accuracy than the plot line for point-level constraints only.

During testing, it was found that the point-level constraints are most useful in initial active learning iterations for “coaxing” points into a nearly adequate segmentation quickly. This is effective if the points’ feature vectors contain attributes that can distinguish the objects being segmented and “coaxed” apart or together. If the points’ feature vectors do not contain attributes that can distinguish two objects as being together or apart, then the cluster-level constraints are much more effective, as then, the human user can quickly specify combinations of relationships between whole groups of points without doing this on a tedious, point-by-point basis. During testing, it was found that the fences were the hardest items to separate in the segmentation. Hence, many of the constraints were for the fences. This would suggest that the feature vectors used did not have the right attributes for segmenting fences. More contextual attributes may remedy this, as mentioned in the Future Work section.

Figure 5.9 qualitatively shows the higher level of accuracy achievable with both cluster-level and point-level constraints. The figure shows the same training patches as in Figure 5.3. For each, it shows an optical image of the area for comparison, the best segmentation
CHAPTER 5. RESULTS

(a) Training Patch 1

(b) Training Patch 2
Figure 5.8: Chart of Training Patches’ Accuracy Improving as Cluster-Level and Point-Level Constraints are Added With Active Learning to Guide Selection of Points and Clusters for Constraints (Square Data Points) Compared to Point-Level Constraints Only Being Added With Active Learning (Diamond Data Points)
achieved with point-level constraints only, and then the higher-quality segmentation with cluster-level constraints as well.

As listed in Chapter 3 on related work, others, such as Stumpf, Lachiche, Malet, Kerle, and Puissant \cite{Stumpf+2012}, have already shown that batch selection and labelling of a cohesive geographic area of remote sensing data is more efficient (in number of iterations and in total human user time) for active learning for classification than the labelling of single points in disparate geographic areas, requiring a repeated re-assessment of context for the human user.

Videos of 3D visualizations of the training patches and other additional materials are available at http://sites.google.com/site/activelearninglabellinglidar.
CHAPTER 5. RESULTS

Figure 5.9: Training Patches Comparison of the Best Segmentation With Only Point-Level Constraints (Second Row) Versus With Both Cluster-Level and Point-Level Constraints (Third Row) (Optical Image Displayed in the Top Row and Underlaid Beneath the Other Images.) LIDAR and optical data are from Leading Edge Geomatics [Lea13].
Chapter 6

Conclusion

Using two types of semi-supervised, active learning, embedded in each other’s iterations, improves the speed and accuracy of creating and labelling a minimal set of training data, just enough for an accurate classification.

Using active learning iterations for the segmentation and labelling of airborne LIDAR data to make training data by employing constrained hierarchical clustering with a enhanced L-method for selecting the ideal number of clusters speeds the creation of the training data. The user interactively uses his/her expertise, with software guidance, to refine the accuracy of the segmentation in key areas. It is more efficient than semi-supervised learning with random selection of points for applying constraints. It is more accurate than segmentation without constraints.

Using active learning iterations for classification to add more training data in key areas of low confidence in the predicted results expedites the creation of only the minimum amount of training data needed to have a high-accuracy result.

Batch selection of LIDAR data points speeds up the active learning for segmentation over the selection of single points for applying constraints. Batch selection of LIDAR data points speeds up the active learning for classification over labelling of single points in disparate geographic areas, requiring a re-assessment of context for the human user.

6.1 Future Work

Future work could add on to the work in this thesis in several ways — through the infrastructure, in regards to the segmentation, in the development of the feature vectors, in regards to the classification, and with the test data.
Infrastructure:

The infrastructure could be improved, and likely the speed of the software increased, by developing a 3D indexing system for accessing the point cloud data more quickly, such as the one that was suggested by Schön, Mosa, Laefer, and Bertolotto [Sch+13].

Semi-Supervised, Active Learning for Segmentation:

The semi-supervised, active learning for segmentation could be made more flexible by allowing a more complex specification of the cluster-level, should-not-link constraints’ splitting location. The split being specified as simply a set of planes is too simple for the complex geometry that can be found in an urban LIDAR point cloud. Specifying the splitting location with a mesh would be more flexible, allowing more complex splitting geometries, though this would add the to computation time in applying the constraint.

More experimentation could be done with other constrained clustering algorithms, comparing the results and performance. In particular, experimentation with active constrained spectral clustering [WD10], as was used by Wang et al. [Wan+12], may allow use of all of the possible attributes in the feature vector for segmentation by clustering (instead of the subset that is used now) because of the dimensionality reduction that is intrinsic in the spectral clustering.

Alternatively, feature selection and weighting could be incorporated into the active learning iterations. The attributes could be added or removed, and the weights adjusted, based on the user adding should-link or should-not-link constraints. After a constraint is added, the algorithm would look for which attributes would encourage or discourage those clusters to be placed separately or together. [BDW08]

Attributes in the Feature Vectors:

Supplementing the feature vector with additional attributes could help some of the more challenging objects to be distinguished more clearly. Attributes based on context could help distinguish a basketball hoop pole from a street light, for example. It could also help where the resolution of the data causes contiguous objects to not appear contiguous in the point cloud. Noting nearby, non-contiguous points and analysing the context of the area could help distinguish attributes that they share, indicating that they should be segmented together. In particular, this may help with fences.
CHAPTER 6. CONCLUSION

Finding the curve skeletons of the contiguous neighbourhoods of points could be used to infer other more complex attributes (orientation, shape, topology, etc.) if the processing is fast enough for the active learning interactivity. [Cao+10]

Semi-Supervised, Active Learning for Classification:

The semi-supervised, active learning for classification could be made more automatic by having the algorithm select the next patch of training data for segmenting and labelling, based on the uncertainty of the classification from the SVM. Unfortunately, this would force the user to select a particular patch; whereas, in the current thesis work, an uncertainty map is shown to guide the user to select particular areas, but the patch is not forced/selected for them.

More experimentation could be done with the selection metric for the active learning iterations, too, such as incorporating diversity in the criteria, in addition to the uncertainty.

Test Data:

Experimentation with true 3D “vegetation and other objects” analysis results as the test data inputs for this thesis work would validate the theoretical, full 3D implementations done in the this thesis work.

Also, experimenting with full waveform LIDAR data and building new attributes for feature vectors from this expanded data would be interesting, as has been considered by some other authors, such as Mallet, Soergel, and Bretar [MSB08].

Adding ground-based LIDAR scans to the point clouds, in addition to the airborne data, would likely allow vertical objects to be classified more accurately, adding more points on the vertical face to objects like fences, street light posts, and traffic light poles. Experimentation with this addition would be interesting in future work.
Bibliography


Appendix A

Software Requirements and High-Level Design

This semantic labelling software component is referred to as the “template matching” software component in the appendices, to reflect the naming convention used in the software code base. Similarly, labelled segments that will be used for training data are referred to as “templates” in these appendices. The semantic labelling/template matching software component is an enhancement to an existing software product called Feature Type Interpreter (FTI) that is developed by Object Raku Technology Incorporated [Obj13]. The FTI software product is a PC application with a Windows-based GUI. A screen capture of the main display of the software is shown in Figure A.1.

The existing FTI software already provides the following services that are related to or utilized by this semantic labelling/template matching enhancement:

- Reads airborne LIDAR point cloud files in LAS file format.
- Resamples the points to a regular grid.
- Converts the 3D data to two sets of 2.5D data, one for the first return and one for the last return, making a Digital Surface Model (DSM) for each of the elevations of the Earth and the objects on it.
- Displays the elevation of the LIDAR signal’s first return DSM, last return DSM, the intensity of the LIDAR signal on the last return, and an associated colour image, if available.
Figure A.1: Main Display of Object Raku Technology’s Feature Type Interpreter [Obj13]

- Estimates the bare-Earth/ground to make a DTM of the Earth without any objects on it.
- Detects the buildings in the DSM.
- Classifies all non-ground and non-building points as vegetation.
- Additional analyses detect roads and watercourses.
- Saves the analysis results as geoTIFF files and displays them in the FTI main window.
- Exports an updated version of an LAS file with classifications.

This new semantic labelling/template matching software component adds the following additional services to the FTI software:

- Allows the user to select a small geographical area to be a training patch.
- Segments the data points that were previously classified as vegetation in that training patch by clustering the data points using user-selectable and tunable attributes.
• Automatically estimates the ideal number of segments in the training patch.

• Allows the user to view the results of the segmentation of the training patch.

• Identifies segments that are near to the decision boundary to direct the user’s attention there (active learning).

• Allows the user to specify should-link and should-not-link constraints on points or segments to improve the segmentation with this human semi-supervised guidance.

• Lets the user label each segment. Each of these labelled segments is a “Template” to use for “Template Matching”.

• Uses the labelled segments/“Templates” from the training patch to train a SVM, which then is used to predict the classification of the other data points beyond the training patch.

• Displays the classification results on the main FTI screen and saves them to geoTIFF files.

• Identifies data points that had a low probability on the assigned classification to direct the user’s attention there (active learning).

• Displays the a greyscale indication of the associated probabilities of the assigned classifications on the main FTI screen and saves it to a geoTIFF file.

• Allows the user to add training patches on areas of low probability/confidence to define additional training templates on those areas, and then, to re-run the classification with this set of training data that has been improved with some human semi-supervised guidance.

A.1 Requirements

The following requirements define the intended behaviour and performance of the new template matching enhancement:

• Define templates from the results of the existing vegetation analysis.

• The templates shall distinguish human-made objects separately from vegetation.
- Utilize these templates to find similar objects to those templates in a data set that is the result of the existing vegetation analysis.

- The algorithm should execute at a speed that allows a user on a high-performance PC to complete a Template Matching Analysis within a regular workday or overnight.

- The settings used to create the templates shall be saveable for work in a future session with the FTI software.

- The templates shall be able to be refined by the user’s semi-supervised guidance in an active learning framework.

- The templates shall be saveable and then re-loadable for work in a future session with the FTI software, possibly on different but similar LIDAR data sets.

- The classification results, that is the “template matches”, shall be saved in geoTIFF files and displayed in FTI.

- The classification shall be able to be re-run once some additional templates are defined, or existing templates refined, to attempt to improve the classification results. The additional templates shall be added with the user’s semi-supervised guidance in an active learning framework.

The design of the new template matching enhancement considers the following design goals in directing decisions in the design:

- The algorithm should be generic enough to work on other template and data set inputs, rather than the vegetation analysis result only, to allow for future enhancement. For example, a future enhancement may call for the use the full DSM as the input to detect building templates as well.

- The design should allow the processing of true 3D point clouds instead of 2.5D data if it is provided as the input data instead.

- The design should leverage the ability of human users’ natural human visual acuity and discretion with visualizations of the data in the active learning iterations.
A.2 Software Architecture

The software component adds on to the existing FTI software with minimal disruption to the existing architecture. Figure A.2 shows the context of the new software component with the existing software, the decomposition of the new software component into its modules, and the interfaces of those modules with the existing FTI software.

The new software component is grouped into the following new modules:

- Template Matching Graphical User Interface
- Template Creation Attributes and Weights
- Templates
- Template Matching Thread Execution Manager
- Template Matching Dynamic-Link Library External Interface
- LIDAR Point Cloud
• Support Vector Machine for a LIDAR Point Cloud

• Utilities

Note that most of the new code is contained in a new dynamic-link library (DLL). This is to keep the design modular. It decouples the tools and programming language used for the Template Matching Component from the main FTI program. Alternative designs were to completely add the new component inside of the existing FTI software or to add the new component to one of the existing DLL’s. For modularity and independence, the separate Template Matching DLL design was selected.
Appendix B

Software Detailed Design

A more detailed description of each of the modules that were outlined in Appendix A follows.

B.1 Template Matching Graphical User Interface

The Template Matching GUI allows a regular user to interactively guide the creation of templates, label the selected templates, use the templates to search for matches in the rest of the geographical area under study, and then continue to refine the templates to continue to improve the classification, using active learning for both segmentation to make the templates and in the classification iterations to add, remove, or delete training patches.

A regular user may select a named set of parameters to use for creating templates. A more advanced user, if given access to the privileged options, could have access to the details for that named set of parameters. These options are available via a button and an additional pop-up dialogue box. The button is hidden for normal users but may be turned on with a flag to indicate that this is a privileged user.

The privileged user has access to view and edit the detailed internal parameters that include the attributes that make up the feature vector that is used for the segmentation to build the potential templates. These internal parameters include the following:

- Which attributes/features of the data are considered for determining the potential templates
- The relative weight assigned to each of these attributes/features
• The horizontal distance threshold inside of which is considered to contain the immediate spatial neighbours of a particular data point

• The horizontal and vertical distance thresholds that are used to determine which other data points are considered to be adjacent/contiguous with a particular data point

• The horizontal distance threshold inside of which to consider context, as a place-holder for future work (Section 6.1).

Figure B.1 highlights the part of the Template Matching GUI for selecting these parameters. This is the view for privileged users, which is the same as the one for regular users, except for the addition of the “Details” button. Figure B.2 shows the various tabs of the dialogue box for the privileged users that is displayed by clicking on the “Details” button. Privileged users can view and select attributes/features in the tabs of this dialogue box and load and save the attributes/features from previous sessions. There is a tab for point-based attributes named “Basic”, immediate-neighbour-based attributes named “Using Immediate Neighbourhood”, and contiguous-neighbour-based attributes named “Using All Contiguous Neighbours”. There is also a tab for attributes that are based on context named, “Using Region of Context”, but this is simply a place-holder for future work (Section 6.1).

After the user has selected the internal parameters for finding the templates (either implicitly for a regular user or in more detail for an expert/privileged user), the user can select which small geographic area should be used as the training patch for making templates. Figure B.3 highlights the button on the Template Matching GUI for starting selection of a new training patch.

The software then automatically segments the selected training patch and estimates the ideal number of segments in that training patch. The potential list of templates is displayed to the user in the Template Matching GUI. Each potential template is displayed both in the image of the training patch and as an item in a list box. The line item in the list box and the geographic areas in the image of the training patch for the potential template item are both coloured the same colour for the human user to be able to correlate them. Figure B.4 shows this interface.

The user may assign a textual label for a template, if that template is a relevant object. If it is not a template item of interest, then it may be deleted from the list of templates.
Figure B.1: Template Matching Graphical User Interface for Specifying Parameters for Template Creation

Figure B.2: Template Matching Graphical User Interface Dialogue Box for the Privileged Users for Specifying Parameters for Template Creation
Figure B.3: Template Matching Graphical User Interface for Selecting a Training Patch for Template Creation
Figure B.4: Template Matching Graphical User Interface for the Template Items
The user may also select additional geographic areas for training patches, using the same or a different set of parameters for finding the templates, and, again, label the templates of interest.

The user may save or load the list of labelled templates for work in a future FTI software session by using the “Export” and “Import” buttons.

Figure B.4 also highlights the buttons for labelling, deleting, importing, and exporting templates.

If the segmentation that created the potential templates is not yet satisfactory to the user, then the user may improve them, using active learning by adding cluster-level or point-level constraints to be applied to the clustering for segmentation. The user may select to merge two or more templates using the “Merge” button. Alternatively, the user may select to split one or more templates along a set of planes by using the “Split” button and then, specifying a series of planes by right-clicking the mouse to continue adding planes and then, left-clicking the mouse to finish adding the last plane. The user may click the “Single-Points” “Select Points for Should-Merge” button and select two or more single points that should be merged in the same template. The user may click the “Single-Points” “Select Points for Should-Be-Split” button and select two or more single points that should each be in a different template. Multiple points are specified by using a right-click of the mouse to continue selecting and a left-click of the mouse to finish selecting. After each set of constraints is added, the training patch is re-segmented using those constraints and then re-displayed to the user.

Figure B.4 also highlights the buttons for cluster-level and point-level merging and splitting.

After all of the potential templates have been refined, if needed, and labelled or deleted from the list, the user may select to use these templates to apply to the rest of the geographical areas in the data set, looking for similar items to the template items. Figure B.5 highlights the button for this on the GUI.

After the application of the template matching to all of the geographical areas, the result is displayed in the main window, with the new results added to the list of analysis results on the left of the main window. Figure B.6 illustrates this.

One of the items that is added to the list of analysis results on the left of the main window is a probability result. The probability of the classification is shown as a visualization to the user in the main FTI screen with a grey-scale shown for each point — with lighter values
Figure B.5: Template Matching Graphical User Interface for Applying the Templates to All Areas
being more confident and darker values being less confident. Figure B.6 also shows this in
the list of analyses to the left of the window and Figure B.7 shows the grey-scale probability
visualization.

If the classification is not yet satisfactory to the user, then the user may improve it, using
active learning to add additional training patches, or to improve or to delete existing ones,
in areas where the data points have the highest uncertainty and may have been misclassified.
The user can then re-run the classification with the improved training data.

B.1.1 Classes

The Template Matching Graphical User Interface module partly overlaps with the ex-
isting FTI components. It is noted below which classes are new and which are existing ones
that have updates added to them.

**WP2LidarVisualizationPanel:**

- This class previously existed in the software but was expanded to add functionality
  for template matching in a similar style to the existing types of analyses, such as for
  buildings, vegetation, roads, etc.
Figure B.7: Template Matching Graphical User Interface Results Probability Display

- Responsibilities:
  - Display the named set of parameters.
  - Allow selection of a named set of parameters.
  - Allow selection of a training patch.
  - Display a visualization of the potential templates resulting from the segmentation.
  - Allow adding a splitting or merging constraint, based on clusters or points, to the segmentation for the potential templates.
  - Allow labelling the templates and storing the labels.
  - Allow deletion of potential templates, if they are not “of interest”. Store the information on the deleted potential templates. Any potential templates that are deleted from a training patch that also contains potential templates that have been labelled as “of interest” are kept in a deleted items list to be used as additional “negative examples” in the later machine learning training steps.
  - Export potential templates to a file or import potential templates from a file.
Launch execution of the template matching classification on the full geographic area.

Track which Vegetation Analysis, Bare-Earth Analysis, first and last returns, intensity, and colour sources to use for the Template Matching Analysis.

Maintain the list of training patches.

Allow comparison and deletion of training patches.

Maintain the list of analysis results.

Save and load the workspace with the sets of completed analyses between software working sessions.

Collect and transmit parameters to the Template Matching Thread Execution Manager.

**TemplateMatchingTrainingPatchCustomAttributesDialog:**

- This is a new class.

- Responsibilities:
  
  - Display the attributes and their associated weights, the horizontal distance threshold for immediate spatial neighbours, horizontal and vertical distance thresholds for contiguous spatial neighbours, and horizontal distance threshold for context [a place-holder for future work (Section 6.1)] — the parameters that define a named set of parameters for generating potential templates in a training patch.
  
  - Allow editing of these fields.
  
  - Allow saving and loading of the attributes, weights, and other parameters defining a named set of parameters.

**B.2 Template Creation Attributes and Weights**

This new Template Creation Attributes and Weights module stores values for the sets of named parameters that can be used to segment data points in the training patch in order to display potential templates.
The module stores a list of names to define various sets of parameters that could be optimized for finding different objects to be templates — cars, vegetation versus human-made objects, etc. For each named set of parameters, the module stores the list of attributes, weights, and thresholds that should be used for that named type of analysis.

If a user changes any parameters in a named set of parameters, a copy of the adjusted parameters, with a new, unique name, is stored in this Template Creation Attributes and Weights module.

Also, after the parameters have been used to create a set of potential templates, if the parameters had to be changed during the generation of those potential templates, e.g. the file of intensity values was actually unavailable, so intensity attributes were not able to be used in the feature vector, then an “as-run” version of the set of parameters with a new, unique name, is also stored in this module. This enables the later viewing of the attributes and other parameters of a previously-generated training patch’s templates.

The data stored in this module for each named set of parameters mirrors that which is displayed by the Template Matching Custom Attributes dialogue box displayed in Figure B.2.

The set of attributes that may be selected and weighted is stored in this module. They fall into three categories:

- **Point-Based Attributes:**
  - x location
  - y location
  - z location of the LIDAR signal’s first return
  - z location of the LIDAR signal’s last return
  - height above the z-value set as ground for the LIDAR signal’s first return
  - height above the z-value set as ground for the LIDAR signal’s last return
  - intensity of the LIDAR signal’s last return
  - associated colour’s red value
  - associated colour’s green value
  - associated colour’s blue value
  - number of LIDAR signal returns for this point
  - difference in z-value between the first and last LIDAR signal return
• Immediate-Neighbour-Based Attributes:
  
  – number of immediate neighbours around this point
  
  – range of z-values of the LIDAR signal’s first return in the immediate neighbourhood
  
  – range of z-values of the LIDAR signal’s last return in the immediate neighbourhood
  
  – range of z-values above the defined z-value for ground of the LIDAR signal’s first return in the immediate neighbourhood
  
  – range of z-values above the defined z-value for ground of the LIDAR signal’s last return in the immediate neighbourhood
  
  – normal of the immediate neighbourhood using the LIDAR signal’s first return:
    ◦ x-component
    ◦ y-component
    ◦ z-component
  
  – normal of the immediate neighbourhood using the LIDAR signal’s last return:
    ◦ x-component
    ◦ y-component
    ◦ z-component
  
  – angle to the horizontal of the normal of the immediate neighbourhood using the LIDAR signal’s first return
  
  – angle to the horizontal of the normal of the immediate neighbourhood using the LIDAR signal’s last return
  
  – “smoothness”/planar error of the immediate neighbourhood using the LIDAR signal’s first return
  
  – “smoothness”/planar error of the immediate neighbourhood using the LIDAR signal’s last return
  
  – average z-value of the immediate neighbourhood of the LIDAR signal’s first return
  
  – average z-value of the immediate neighbourhood of the LIDAR signal’s last return
  
  – average height above the z-value set as ground of the immediate neighbourhood of the LIDAR signal’s first return
– average height above the z-value set as ground of the immediate neighbourhood of the LIDAR signal’s last return
– average intensity of the immediate neighbourhood of the LIDAR signal’s last return
– average number of LIDAR signal returns of the immediate neighbourhood
– average difference in z between the first and last LIDAR signal returns of the immediate neighbourhood
– average number of immediate neighbours
– average normal between all the points in the immediate neighbourhood using the LIDAR signal’s first return:
  ◦ x-component
  ◦ y-component
  ◦ z-component
– average normal between all the points in the immediate neighbourhood using the LIDAR signal’s last return:
  ◦ x-component
  ◦ y-component
  ◦ z-component
– average angle to the horizontal of the normal between all the points in the immediate neighbourhood using the LIDAR signal’s first return
– average angle to the horizontal of the normal between all the points in the immediate neighbourhood using the LIDAR signal’s last return
– average “smoothness”/planar error of all the points in the immediate neighbourhood using the LIDAR signal’s first return
– average “smoothness”/planar error of all the points in the immediate neighbourhood using the LIDAR signal’s last return

• Contiguous-Neighbour-Based Attributes:
  – membership in the contiguous neighbourhood
  – number of contiguous neighbours connected to this point
- range of z-values of the LIDAR signal’s first return in the contiguous neighbourhood
- range of z-values of the LIDAR signal’s last return in the contiguous neighbourhood
- range of z-values above the defined z-value for ground of the LIDAR signal’s first return in the contiguous neighbourhood
- range of z-values above the defined z-value for ground of the LIDAR signal’s last return in the contiguous neighbourhood
- normal of the contiguous neighbourhood using the LIDAR signal’s first return:
  - x-component
  - y-component
  - z-component
- normal of the contiguous neighbourhood using the LIDAR signal’s last return:
  - x-component
  - y-component
  - z-component
- angle to the horizontal of the normal of the contiguous neighbourhood using the LIDAR signal’s first return
- angle to the horizontal of the normal of the contiguous neighbourhood using the LIDAR signal’s last return
- “smoothness”/planar error of the contiguous neighbourhood using the LIDAR signal’s first return
- “smoothness”/planar error of the contiguous neighbourhood using the LIDAR signal’s last return
- average z-value of the contiguous neighbourhood of the LIDAR signal’s first return
- average z-value of the contiguous neighbourhood of the LIDAR signal’s last return
- average z-value above the defined z-value for ground of the contiguous neighbourhood of the LIDAR signal’s first return
– average z-value above the defined z-value for ground of the contiguous neighbourhood of the LIDAR signal’s last return
– average intensity of the contiguous neighbourhood of the LIDAR signal’s last return
– average number of LIDAR signal returns of the contiguous neighbourhood
– average difference in z between the first and last LIDAR signal returns of the contiguous neighbourhood
– length of the largest dimension of the contiguous neighbourhood using the LIDAR signal’s first return — place-holder for future work (Section 6.1)
– length of the largest dimension of the contiguous neighbourhood using the LIDAR signal’s last return — place-holder for future work (Section 6.1)
– direction of the largest dimension of the contiguous neighbourhood using the LIDAR signal’s first return — place-holder for future work (Section 6.1)
– direction of the largest dimension of the contiguous neighbourhood using the LIDAR signal’s last return — place-holder for future work (Section 6.1)
– length of the smallest dimension of the contiguous neighbourhood using the LIDAR signal’s first return — place-holder for future work (Section 6.1)
– length of the smallest dimension of the contiguous neighbourhood using the LIDAR signal’s last return — place-holder for future work (Section 6.1)
– direction of the smallest dimension of the contiguous neighbourhood using the LIDAR signal’s first return — place-holder for future work (Section 6.1)
– direction of the smallest dimension of the contiguous neighbourhood using the LIDAR signal’s last return — place-holder for future work (Section 6.1)

• Context-Based Attributes:
  – place-holder for future work (Section 6.1)

As explained in Section 4.1.6, all of these attributes are assigned to a single point’s feature vector, but those in categories two through four are calculated in a transformed space that considers its neighbours for calculating these attributes that relate to its context in the spatial, geographical scene.
The other parameters stored in this module for each named set of parameters are the following:

- The horizontal distance threshold inside of which is considered to contain the immediate spatial neighbours of a particular data point
- The horizontal and vertical distance thresholds that are used to determine which other data points are considered to be adjacent/contiguous with a particular data point
- The horizontal distance threshold inside of which to consider context, as a place-holder for future work (Section 6.1).

Section 4.1.6 defines these parameters more specifically.

B.2.1 Classes

**TemplateMatchingAttributes:**

- Responsibilities:
  - Store the data for the attributes, associated weights, and other parameters for the list of named sets of parameters.
  - Save a named sets of parameters to a file.
  - Load a named sets of parameters from a file.

B.3 Templates

This new Templates module stores values for the list of templates that results from segmenting data points in training patches.

The module stores a list of templates’ data and also a list of sources data for those templates, along with their list of any constraints that have been (and will continue to be) applied to the templates from that source.

The data values stored in this module for each cluster/segment/template are the following:

- identification number within its source, e.g. 1, 2, 3, ...
The data values stored in this module for each source of templates are the following:

- unique identifier
- friendly name, e.g. “TP 2”
- input file name and path for the bare-Earth
- input file name and path for the first return
- input file name and path for the last return
- input file name and path for the intensity of the last return
- input file name and path for the colour
- input file name and path for the mask
- outputted file name and path for the training patch segmented result
- x, y, width and height of the training patch
- top left and bottom right map co-ordinates of the training patch
- named set of parameters used
- list of constraints, for which the following is stored for each:
  - true for should-link/“merge” or false for should-not-link/“split”
  - list of x, y, and z-values of single points or list of template/cluster/segment identification numbers that are involved in the constraint
  - list of planes of split (applicable only to cluster-level, should-not-link constraints)
APPENDIX B. SOFTWARE DETAILED DESIGN

The active learning and semi-supervised constraints are explained in Chapter 4’s section on Semi-Supervised, Active Learning for Segmentation and Labelling To Create Training Data. The source files are explained in more detail later in this appendix.

B.3.1 Classes

TemplateMatchingTemplates:

- Responsibilities:
  - Store the data for the templates and associated sources, including the constraints.
  - Allow templates to be added and deleted.
  - Allow sources to be added and deleted.
  - Allow constraints to be added to a source.
  - Allow a template to have its label updated.
  - Organize the data for use in training the machine learning model.
  - Organize the data for displaying in a list box.
  - Retrieve templates only for a particular source.
  - Delete the templates from a particular source.
  - Retrieve a distinct list of all of the labels used.
  - Save the templates and sources, including constraints, to a file.
  - Load the templates and sources, including constraints, from a file.

B.4 Template Matching Thread Execution Manager

This new Template Matching Thread Execution Manager is the interfacing module between the main FTI program and the new Template Matching DLL.

The Template Matching Thread Execution Manager receives the set of parameters that is needed for the DLL, stores them during the execution of the thread, translates them to the format that is needed by the DLL, and then launches and monitors execution of the DLL. After the execution of the DLL has returned, the Template Matching Thread Execution Manager receives the results of the DLL execution and then translates them to the formats that are needed by the main FTI program.
Consolidating the interface between the main FTI program and the DLL allows any format translations and work-arounds needed for language dependencies to be isolated and controlled.

B.4.1 Classes

CaptureTemplateMatchingThreadWithState:

- Responsibilities:
  - Receive the parameters needed for the execution of the DLL.
  - Store the parameters given to the DLL.
  - Translate the parameters needed for the execution of the DLL from the format used in the main FTI program into the format needed by the DLL.
  - Execute the DLL.
  - Upon completion of the DLL’s execution, receive the results from the DLL.
  - Translate these results to the formats needed by the FTI main application.

B.5 Template Matching Dynamic-Link Library External Interface

The Template Matching DLL External Interface defines the services that the DLL provides to the software that interfaces with it. The Template Matching DLL has two services. One is an interface for segmenting data points of a training patch into potential templates. The second is an interface for using labelled templates to train a machine learning model, and then using that trained model to classify the data points in a larger geographic area.

B.5.1 Interface for Segmentation into Templates

This interface allows programs to call the DLL, providing the required inputs, and, in return, to receive a list of potential templates.
Functions

identifyTemplateMatchingTrainingClustersInGeotiffFile:

- Responsibilities:
  - Receive the inputs from the calling program, FTI.
  - Translate the parameters received in the interface from the calling program into the format needed by the objects used in the DLL.
  - Create a LIDAR Point Cloud Object.
  - Invoke the population of the LIDAR Point Cloud with the specified geoTIFF files.
  - Ask the LIDAR Point Cloud to update the list of requested attributes to reflect which attributes can actually be used, according to the available data.
  - Request the LIDAR Point Cloud to return a version of its data points in a specified geographical area, segmented into potential templates by utilizing the specified attributes and weights. The LIDAR Point Cloud also returns the estimated ideal number of segments in that specified geographical area, the number of points in each segment, and some guiding active learning comments.
  - If requested by the calling program, ask the LIDAR Point Cloud to write its proposed segmentation of the specified geographical area to a geoTIFF file.
  - Translate the parameters used by the objects in the DLL into the format needed to return the output parameter values to the calling program.
  - Return the outputs to the calling program, FTI.
  - Create a status window with a progress bar that is used throughout the execution of the function and destroy it at the end.

- Inputs from the calling program:
  - Path and file name for a geoTIFF file of the bare-Earth z-values that were calculated previously in the Ground-Only/Bare-Earth Analysis. (Note that a pixel in this geoTIFF file with the value of -99999.0 means “no data in this pixel”.)
− Path and file name for a geoTIFF file of the z-values of the first LIDAR signal return. (Note that a pixel in this geoTIFF file with the value of -99999.0 means “no data in this pixel”.)

− Path and file name for a geoTIFF file of the z-values of the last LIDAR signal return. (Note that a pixel in this geoTIFF file with the value of -99999.0 means “no data in this pixel”.)

− Path and file name for a geoTIFF file of the intensity value of the last LIDAR signal return. (Note that a pixel in this geoTIFF file with the value of 0.0 means “no data in this pixel”.)

− Path and file name for a geoTIFF file of a colour image to additionally associate with the LIDAR data.

− Path and file name for a geoTIFF file to use as a mask to indicate which x, y-values in the other input geoTIFF files should be read and kept. It must be an RGB colour geoTIFF file. Any pixel values that are non-white (i.e. not a value of Red: 255, Green: 255, Blue: 255) are interpreted to be requested for keeping, using, and analysing. The current use of the DLL has the geoTIFF file that is the result of the FTI Vegetation Analysis as the file used in this field. This requests that the DLL only analyse the pixels that have been classified as vegetation, effectively removing the pixels classified as ground and buildings from the set under analysis. A different file could be used as the mask file for future enhancements.

− Path and file name for the desired location of the RGB geoTIFF output file of the potential templates. Each potential template is coloured with a different colour in the output file.

− Start x and y, width, and height pixel values for defining the small training patch sub-area of interest to be extracted from the input geoTIFF files.

− Identifiers to indicate which attributes of the data should be used for the segmentation. For each of these attributes, there is also a weighting specified for the relative importance of each of these attributes. These were the values that were originally from the Template Creation Attributes and Weights module.
− Horizontal distance threshold for defining which other points are considered to be immediate spatial neighbours of a particular data point. (This value is usually set to be approximately the resolution of a pixel by the calling program.)

− Horizontal and vertical distance thresholds for defining which other data points are considered to be contiguous with a particular data point. (The horizontal value is usually set to be approximately the resolution of a pixel by the calling program. The vertical value is usually set to be approximately one metre by the calling program.)

− Horizontal distance threshold for defining which other data points are considered to be within a region of context of a particular data point. This is a place-holder for future work (Section 6.1).

• Outputs to the calling program:

  − The RGB geoTIFF output file of the potential templates. The file is named using the requested file name and path. Each potential template in the file is coloured with a different colour.

  − An estimate of the ideal number of potential templates in the small training patch sub-area of interest.

  − An updated set of the attribute identifiers for which attributes of the data were actually used (according to the available data) for the segmentation.

  − A list of the RGB colour that corresponds to each potential template that was used in the output geoTIFF file.

  − A list of the number of data points that there are in each potential template.

  − A list of any active learning guidance comments for any of the potential templates.

**identifyTemplateMatchingTrainingClustersInLASFile:**

• Place-holder for possible future enhancement.

• Intended to have the same responsibilities as the “identifyTemplateMatchingTrainingClustersInGeoTIFFFile” function, but instead of loading the data from 2.5D geoTIFF files, it loads it from 3D LAS files.
**reIdentifyTemplateMatchingTrainingClustersInGeotiffFile:**

- **Responsibilities:**
  - Same as for the “identifyTemplateMatchingTrainingClustersInGeotiffFile” function, except for the request of the LIDAR Point Cloud to return a version of its data points in a specified geographical area. In this function, the LIDAR Point Cloud is requested to re-segment the specified geographical area into potential templates by utilizing the specified set of constraints in addition to the specified attributes and weights, re-using the same segment colours as last time, as much as possible.

- **Inputs from the calling program:**
  - Same as for the “identifyTemplateMatchingTrainingClustersInGeotiffFile” function
  - Path and file name for the geOTIFF file that was the former segmentation, such as the geOTIFF file of the training patch’s last segmentation result. It must be an RGB colour geOTIFF file.
  - A list of the RGB colours that corresponds to each potential template in the former segmentation geOTIFF file.
  - A list of labels (may be just “TBD”) that corresponds to each potential template in the former segmentation geOTIFF file.
  - A list of true for should-link/“merge” or false for should-not-link/“split” for each constraint.
  - A list of constraints’ lists of x, y, and z-values of single points that are involved in the constraint.
  - A list of constraints’ lists of template/cluster/segment identification numbers that are involved in the constraint.
  - A list of constraints’ list of planes of split (applicable only to cluster-level, should-not-link constraints).
• Outputs to the calling program:
  
  − Same as for the “identifyTemplateMatchingTrainingClustersInGeotiffFile” function.
  
  − An updated list of labels (may be just “TBD”) that corresponds to each potential template in the new segmentation geoTIFF file.
  
  − An updated list of constraints’ lists of template/cluster/segment identification numbers that are involved in the constraint.

**reIdentifyTemplateMatchingTrainingClustersInLASFile:**

• Place-holder for possible future enhancement.

• Intended to have the same responsibilities as the “reIdentifyTemplateMatchingTrainingClustersInGeotiffFile” function, but instead of loading the data from 2.5D geoTIFF files, it loads it from 3D LAS files.

**B.5.2 Interface for Classification for Finding Template Matches by Training and Predicting**

This interface allows programs to call the DLL, providing the required inputs, and, in return, to receive the classified data points.

**Functions**

**identifyTemplateMatchesUsingGeotiffFiles:**

• Responsibilities:

  − Receive the inputs from the calling program, FTI.
  
  − Translate the parameters received in the interface from the calling program into the format needed by the objects used in the DLL.
  
  − Create a SVM for LIDAR Point Cloud Object.
  
  − Invoke the training of the SVM.
  
  − Request the SVM to predict the classifications of the requested data.
  
  − Request geoTIFF files to be written for the labelled classifications.
Create a status window with a progress bar that is used throughout the execution of the function and destroy it at the end.

- Inputs from the calling program:
  - For the training of the machine learning model:
    - A list of labelled template items. Each template item has the following data:
      - A textual label.
      - An identifier for the set of source data from which it comes.
      - Its representative RGB colour in the potential templates geoTIFF file of that source data.
      - Templates may be included just to be negative examples, i.e. ones that are not of interest. They should be given the textual label of “NotInterested” in the calling program.
    - A list of data sources outlining the inputs from which the various templates originate, so that data can be re-loaded as-needed for the machine learning training. (For example, each training patch would be one source data entry in this list. It has been named source data instead of training patch because the labelled templates could be loaded from a file rather than being generated in each software session through training patches.) Each source data item has the following data:
      - Path and file name for the original input files that provided the data for making those templates:
        - The geoTIFF file of the bare-Earth z-values that were calculated previously in the Ground-Only/Bare-Earth Analysis.
        - The geoTIFF file of the z-values of the first LIDAR signal return.
        - The geoTIFF file of the z-values of the last LIDAR signal return.
        - The geoTIFF file of the intensity value of the last LIDAR signal return.
        - The geoTIFF file of a colour image to additionally associate with the LIDAR data.
        - The geoTIFF file to use as a mask to indicate which x,y-values in the other input geoTIFF files should be read and kept.
APPENDIX B. SOFTWARE DETAILED DESIGN

* Path and file name for the RGB geoTIFF output file of the segmented and colour-labelled potential templates.
* Start x and y, width, and height pixel values for the original small training patch sub-area of interest that was extracted from the input geoTIFF files.
* Horizontal distance threshold for defining which other points were considered to be immediate spatial neighbours of a particular data point.
* Horizontal and vertical distance thresholds for defining which other data points were considered to be contiguous with a particular data point.
* Horizontal distance threshold for defining which other data points were considered to be within a region of context of a particular data point. This is a place-holder for future work (Section 6.1).

- Identifiers to indicate which attributes of the data should be used to train the machine learning model. This is unused in the current implementation where all of the attributes are used in training the machine learning model.
  - For the classification:
    - Paths and file names for the input files that provide the data that will be classified:
      * A geoTIFF file of the bare-Earth z-values that were calculated previously in the Ground-Only/Bare-Earth Analysis.
      * A geoTIFF file of the z-values of the first LIDAR signal return.
      * A geoTIFF file of the z-values of the last LIDAR signal return.
      * A geoTIFF file of the intensity value of the last LIDAR signal return.
      * A geoTIFF file of a colour image to additionally associate with the LIDAR data.
      * A geoTIFF file to use as a mask to indicate which x,y-values in the other input geoTIFF files should be read and kept.
    - Start x and y, width, and height pixel values for defining an area of interest to be extracted from the input geoTIFF files on which the classification will be done. This is typically the full extent of the geographic area defined in the input geoTIFF files, but a sub-area could be specified.
APPENDIX B. SOFTWARE DETAILED DESIGN

◦ Number of pixels to use as buffer around any set of pixels to ensure a good calculation of its neighbours. This is needed because, for the classification, the full geoTIFF file’s area must be processed in pieces, due to PC memory constraints.

◦ Path and root of the file name for the desired location of the RGB geoTIFF output files of the classifications. Each label name, and one more for “Probability”, is appended onto the end of the file to make a set of files of output.

• Outputs to the calling program:
  – The RGB geoTIFF output files of the classifications and the probability visualization.

identifyTemplateMatchesUsingLASFiles:

• Place-holder for possible future enhancement.

• Intended to have similar responsibilities to the “identifyTemplateMatchesUsingGeotiffFiles” function, but instead of loading the data from 2.5D geoTIFF files, it loads it from 3D LAS files.

B.6 LIDAR Point Cloud

This new LIDAR Point Cloud module loads and stores the data related to the LIDAR point cloud and performs calculations on it. Internally, for segmentation by clustering, it uses the open-source “C Clustering Library” [Hoo+04]. It also utilizes several functions from the Utilities module.

B.6.1 Classes

LIDARPointCloud:

• Responsibilities:
  – Read the input geoTIFF files (as listed in more detail in Section B.5): bare-Earth z-values, z-values of the first LIDAR signal returns, z-values of the last LIDAR
signal returns, intensity values of the last LIDAR signal returns, a colour image, and a mask. Functions in the Utilities module are utilized for this. Reading the data from .las files as well is a planned future enhancement.

- Store the following data for each of the point cloud’s data points:
  - x-value
  - y-value
  - z-value of the first LIDAR signal return
  - z value of the last LIDAR signal return (optional)
  - z-value of the bare-Earth/ground
  - intensity value of the last LIDAR signal return (optional)
  - associated red, green, and blue colour components (optional)
  - a cluster identifier (optional)
  - a label (optional)

- Store the co-ordinate system and projection data from the source geoTIFF files to use for any future generation of geoTIFF result files. (Optional only when the data is from a set of geoTIFF files.)

- Store the input-specified distance thresholds for neighbourhood calculations.

- For indexing efficiency, store a raster pointing to the point cloud array indices corresponding to the data point for that pixel location. (Optional only when the data is from a set of geoTIFF files.)

- Calculate derived attributes for the data points.

- Calculate the minimum, maximum, mean, and standard deviation values for the attributes for the data points.

- Calculate a normalized set of the attributes of the data points.

- Find the immediate spatial neighbours of a point, based on the specified distance threshold.

- Find the contiguous neighbourhoods, also known as “blobs”, using the specified distance threshold.

- Calculate the number of contiguous neighbour segments in the point cloud.

- Return a list of all possible attribute identifiers.
− Given a list of attribute identifiers, adjust them to only include the attributes that are useful for segmentation based on the data available.

− Segment the point cloud into segments, based on the requested attributes and requested relative weighting of the attributes. The ideal number of segments is estimated and used for the segmentation.

− Provide some active learning guidance comments on any of the segments, to help with the addition of constraints.

− Count the number of points in each segment/cluster.

− Re-segment the point cloud into segments, based on the requested attributes, the requested relative weighting of those attributes, and any point-level or cluster-level, should-link or should-not-link constraints. The ideal number of segments is estimated and used for the segmentation.

− If previously-separate clusters are being combined during a re-segmentation, update the constraint to have the new, single cluster identifier, instead of both of the previous ones. If previously-combined clusters are being split during a re-segmentation, update the constraint to have both of the new, separate clusters as being involved in the splitting constraint.

− If previously-separate clusters are being combined during a re-segmentation, choose the best textual label between the separate clusters’ former labels, based on majority voting, using the number of points that were labelled with each former label. If previously-combined clusters are being split during a re-segmentation, apply the old textual label to both of the new, separate clusters.

− Reduce the potentially 3D point cloud classifications into a 2D raster of segmented points. Write a geoTIFF file of this raster, using functions in the Utilities module.

B.7 Support Vector Machine for a LIDAR Point Cloud

This new “Support Vector Machine for a LIDAR Point Cloud” module is a SVM supervised machine learning model that has been encapsulated and customized for this application. Internally, it uses the SVM from the open-source LIBSVM Library [CL11]. It also utilizes the LIDAR Point Cloud module and the geoTIFF reading and writing functions from the Utilities module.
APPENDIX B. SOFTWARE DETAILED DESIGN

The “Support Vector Machine for a LIDAR Point Cloud”, using the specified inputs, trains the SVM and then, performs a prediction on the other data specified as input.

B.7.1 Classes

SVMForLIDARPointCloud:

- Responsibilities:
  - Store the supervised machine learning model (a SVM).
  - Pick the distance thresholds to use from the possible various distance thresholds from the different input sources (though usually they are the same in each input source). Store these values to use for calculations later when predicting classifications.
  - Create a LIDAR Point Cloud for each input source.
  - Retrieve the full set of attribute identifiers and reduce/update them according to the available data. Remove the x and y-values which are only locally relevant. Store these values to use for calculations later when predicting classifications.
  - Find the minimum and maximum values for all of the used attributes, between all of the sources, to be able to normalize all of the data points’ attribute values uniformly, regardless of their sources. Store these values to use for calculations later when predicting classifications.
  - Collect the data from each input source and normalize the data values.
  - Collect the corresponding labels for the data points, store the textual labels, and give them a representative index to use with SVM.
  - Perform a grid search to find the best values of $\gamma$ and $C$ to use with the RBF SVM.
  - Train the SVM with the data and corresponding label indices, using the best $\gamma$ and $C$ found.
  - Using a tile-wise buffering system (with some overlap between tiles just for calculating the attribute values), load the input files into the LIDAR Point Cloud to calculate attributes, normalize them, and then use these normalized attributes for having the SVM predict the classifications and, with an enhancement to the SVM, also the probabilities for those classifications.
− Update the status window as the classification progresses.
− Write geoTIFF files of the resulting classifications, using the Utilities functions.
− Write a geoTIFF file for a visualization of the probabilities of the resulting classifications, using the Utilities functions.

B.8 Utilities

This new Utilities module includes many helpful functions used by the other modules. The functions that read and write geoTIFF files use the open-source Libgeotiff [Lib12a] and Libtiff [Lib12b] Libraries.

B.8.1 Functions

readGeoTIFFFileForRaster:

• Responsibilities:
  − Read a geoTIFF file into a layered raster in a 3D vector.
  − Populate a definition of the coordinate system.
  − Populate the Projected Co-ordinate System (PCS) co-ordinates of the four corners of the returned raster.
  − Optionally, return only a sub-area of the geoTIFF.

readTIFFFileForRaster:

• Responsibilities:
  − Read an open TIFF file into a layered raster in a 3D vector.
  − Optionally, return only a sub-area of the geoTIFF.

readRGBATIFFFileForRaster:

• Responsibilities:
  − Read a RGBA TIFF file into a layered raster in a 3D vector.
  − Optionally, return only a sub-area of the geoTIFF.
readGeoTIFFFileForCoordSystemDefinitionAndCornersInPCS:

- Responsibilities:
  - Read a geoTIFF file just for the definition of the coordinate system and/or the PCS co-ordinates of the four corners.
  - Optionally, return only the corners of a sub-area of the geoTIFF.

writeGeoTIFFFileFromRaster:

- Responsibilities:
  - Write a TIFF or a geoTIFF file from a layered raster in a 3D vector.
  - If provided, write the definition of the coordinate system in the TIFF file to make it a geoTIFF file.
  - If provided, write the PCS co-ordinates of the four corners of the returned raster into the geoTIFF file.

writeRGBATIFFFileFromRaster:

- Responsibilities:
  - Write a RGBA TIFF file from a layered raster in a 3D vector.

getIdealNumberOfClustersUsingTheseDistancesAndTheLMethodWithIterativeRefinement:

- Responsibilities:
  - Estimate the ideal number of segments/clusters based on the distances provided, using the L-method and iterative refinement as discussed in Section 4.1.2.
  - Call the “getIdealNumberOfClustersUsingTheseDistancesAndTheLMethod” function.

getIdealNumberOfClustersUsingTheseDistancesAndTheLMethod:

- Responsibilities:
  - Estimate the ideal number of segments/clusters based on the distances provided, using the L-method (no iterative refinement) as discussed in Section 4.1.2.
  - Utilize the “meanSquaredErrorOfTheBestFitLineForThesePoints” function.
**meanSquaredErrorOfTheBestFitLineForThesePoints:**

- Responsibilities:
  - Perform linear regression (finding the best fit line), using the least squares fitting approach, and return the mean squared error of the best fitting result.