Universiteit van Amsterdam

Master Thesis

Delineating linear vegetation elements in agricultural landscapes using LiDAR point clouds

Author:
Chris Lucas
5947332

Supervisors:
Prof. Dr. Ir. W. Bouten
Dr. A.C. Seijmonsbergen

July 23, 2017

Abstract

Intensification of farming and the associated increased size of agricultural fields has caused a major decrease in linear vegetation elements such as tree lines and hedges in rural areas across Europe. These elements are considered being a part of the cultural landscape and functions as corridors in the local ecosystem, hosting many animal and plant species. An inventory of the spatial distribution of these elements is therefore valuable for the conservation of habitats and promoting biodiversity.

Our aim was to develop an automated method to detect these elements using data available nationwide. LiDAR point clouds are increasingly available at these national scales and the 3D view makes them very suitable for detecting vegetation. Geometric properties of the LiDAR points and their neighbourhoods were calculated. Based on the planarity and spherecity properties most of the irrelevant points were removed. To separate the vegetation from the remaining points a random forest classifier was used. A further subdivision into low vegetation and trees was based on the height differences around each return. The vegetation classes were segmented using a region growing method based on a newly developed rectangularity approach. The resulting rectangular regions were then assessed for their linearity by analysing their elongatedness.

The relevant vegetation was segregated well and although small improvements can still be made, those would not result in a better delineation of linear vegetation elements, since the misclassified points are dispersed. Although extraction of irregular vegetation patterns can be optimized, the newly developed method proved to successfully separate the majority of linear and non-linear vegetation elements in the study area, a Dutch agricultural landscape.

Utilizing a high performance computing paradigm the developed method can be used over large areas. The results can be used as an extra indicator in ecosystem and biodiversity assessments. Additionally it can be used in ecological research, for example to refine spatial distribution models of flora and fauna. Furthermore, the method has potential to also extract other elongated objects from point clouds, such as roads and ditches.

Keywords: Linear, Vegetation, LiDAR, Point cloud, Classification, Rectangularity

1 Introduction

Human land use has had a big impact on the structure of landscapes (Turner, 1989). Many landscapes have been divided into patches by human created linear landscape elements (e.g. roads, waterways, tree lines and hedges) (Meyer et al., 2012). This creates a mosaic with a combination of natural and human influenced landscapes of varying sizes and shapes. These patches and their connectivity influence the ecology of the landscape: it effects the distribution of species and the dispersal of seeds, the flow of matter and nutrients, and the water balance (Turner, 1989).

Linear elements, such as roads and railways, form barriers which lead to habitat fragmentation, while others, such as hedgerows and tree lines, can form corridors for animals and plants between patches (Spellerberg and Sawyer, 1999). They also control fluxes, like the flow of water and pollutants via ditches, and by influencing air flow by, for example, tree lines and hedges (Burel, 1996). Furthermore they provide habitats for species who are otherwise not able to live in the agricultural landscape (Spellerberg and Sawyer, 1999).

Agriculture has been an important factor in the creation of linear elements, leading to a mosaic landscape. Linear features like ditches, lynchets and hedgerows are often the result of agricultural practices (Bailly et al., 2008). They have therefore become an important part of the European landscape, which is dominated by agriculture. Intensification of arable farming in Europe has led to larger agricultural fields, simpler crop rotations and a reduction in non-crop features. Consequently the diversity in the landscape and occurrence of linear elements has diminished (Stoate et al., 2001). Thus increasing the importance of the remaining linear features.

Not only do these linear elements have an ecological significance, some also have a cultural value (Jongman, 2004). Linear elements, like tree lines, hedges, waterways and ditches, are often considered traditional and aesthetically pleasing, and are therefore part of natural heritage (Gobster et al., 2007).

Given the significance of linear landscape elements within the European landscape an assessment of the coverage of these elements would be beneficial (Van der Zanden et al., 2013; Bailly et al., 2008). There are multiple ways to approach this. Van der Zanden et al. (2013), for example, used over 200,000 ground observations within the European Union of linear landscape elements and used interpolation to create a map of the density of linear landscape elements in the European Union. This approach can certainly be useful to estimate linear elements in the landscape, but interpolation introduces an

unknown error. A more precise way of assessing the presence of linear elements in the landscape could be identification using remote sensing. This could be done by using high resolution aerial imagery, by using airborne LiDAR (Light Detecting and Ranging), which uses lasers to determine height by measuring the time it takes for the laser to be reflected back (Lefsky et al., 2002; Lim et al., 2003), or by combining these two.

Scanning the earth using lasers leads to Very High Resolution (VHR) elevation data, often with multiple observation points per square meter and thus provides a very detailed 3D image of the landscape (Lefsky et al., 2002). Airborne LiDAR datasets with high point density are becoming increasingly available. Countries like the Netherlands and Denmark, for example, have made LiDAR datasets of the entire country freely accessible (AHN, 2016; SDFE, 2017).

Airborne LiDAR often has two additional properties which are very useful for detecting vegetation: (i) multiple returns, and (ii) intensity of the backscattered signal. When a signal only partly reflects on an object, multiple return values can be recorded (figure 1). Vegetation partly reflects the signal leading to multiple return values, including a last ground return. The shape of vegetation can be extracted using the different return values at a point (Lim et al., 2003). The intensity describes the strength of the returning light (figure 1). This is dependent on the surface on which it reflected and therefore provides meaningful information on the surface origin (Song et al., 2002).

There are generally two main strategies to classify point clouds: (i) create derived raster images (e.g. a DSM and DTM) and use conventional raster based techniques for classification, or (ii) classify the raw point cloud directly using machine learning algorithms (Yan et al., 2015). Using derived raster images for classification is fairly straightforward and can lead to desirable results (Maas, 1999; Charaniya et al., 2004; Brennan and Webster, 2006; Forlani et al., 2006; Antonarakis et al., 2008; Im et al., 2008), but a lot of detail is lost in the conversion from points to raster cells. Therefore directly classifying the point cloud is a more preferred method. Different machine learning algorithms are being tested and used for LiDAR datasets, for example Support Vector Machines (SVM) (Mallet et al., 2008; Zhang et al., 2013; Weinmann et al., 2015), Random Forest (RF) (Chehata et al., 2009; Guo et al., 2011; Niemeyer et al., 2014; Weinmann et al., 2014, 2015), Markov Random Fields (MRF), Conditional Random Fields (CRF) (Niemeyer et al., 2011, 2012, 2014), and Adaboost (Lodha et al., 2007; Weinmann et al., 2015).

After classification of the point cloud the linear objects need to be identified. The identification of linear features has been extensively researched

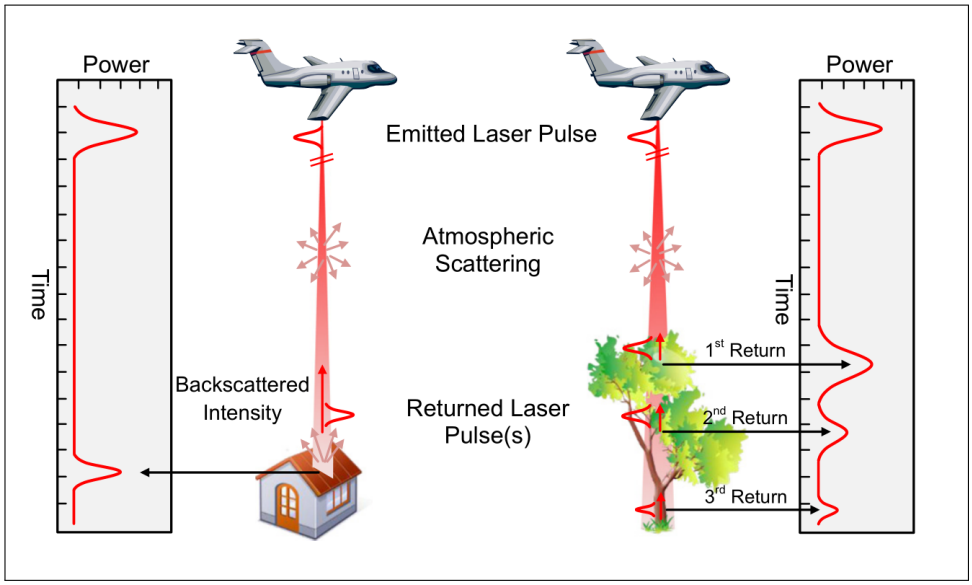

Figure 1: The retrieval of airborne LiDAR data, resulting in backscatters of varying intensities and discrete returns (Yan et al., 2015).

for raster-based imagery. Mainly focussing on the automated delineation of roads from remotely sensed imagery. This process is often done in three steps: (i) edge detection, (ii) road tracking, and (iii) road linking (Quackenbush, 2004). These methods generally work well for roads partly due to the structured nature of a road system. Vegetation is a lot more complex, having lines and patches in all kinds of shapes and sizes. This makes these raster-based techniques difficult to apply to an extraction of linear vegetation elements from points clouds.

In this paper we describe our method to delineate linear objects within vegetation using LiDAR point clouds. To achieve this we use a machine learning algorithm to classify vegetation points based on properties of the LiDAR points and computed neighbourhood parameters. To delineate linear objects within the vegetation points we present a novel way of using a region growing technique to segment the point cloud into rectangular objects. These objects are subsequently checked for linearity based on their elongatedness. The method is validated by comparing the results with manually acquired reference data.

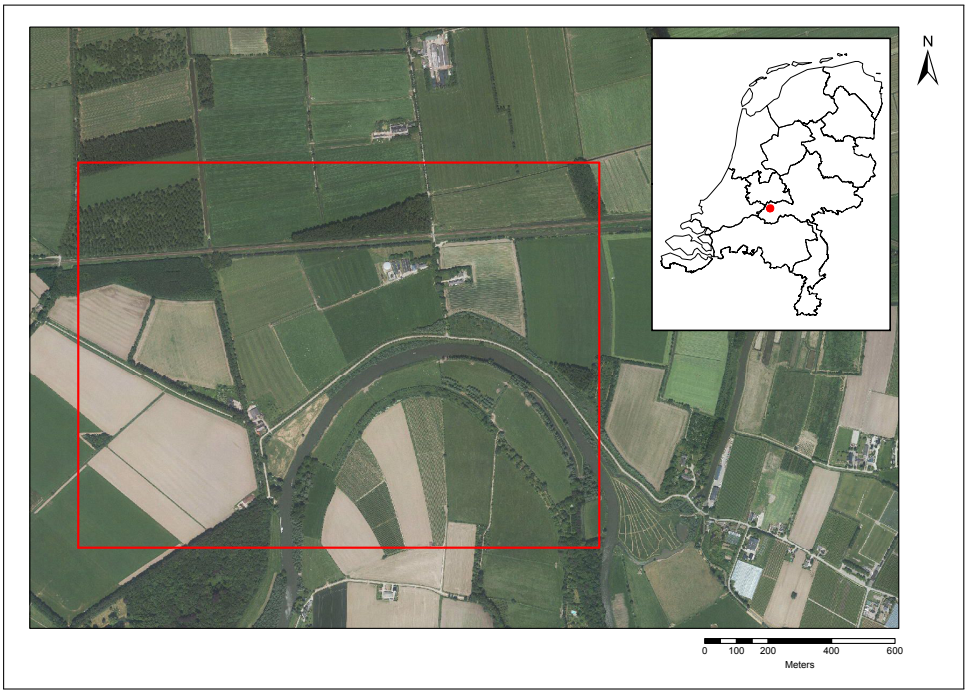

Figure 2: The study area.

2 Study Area

To verify to performance of the developed method these criteria were used for selecting a study area: (i) concerns an agricultural landscape, (ii) contains a variety of linear vegetation elements (straight, curves, corners, attached to non-linear elements), and (iii) contains non-linear vegetation elements. These criteria ensure the both the classification of vegetation and delineation of linear elements can be properly trained and tested.

The chosen study area is located in an agricultural landscape in the middle of the Netherlands between Beesd in the west and Geldermalsen in the east (figure 2). The area is about 1.6 km from east to west and 1.2 km from north to south, spanning an area of almost 2 million square meters.

3 Data

The raw LiDAR point cloud data of this area were retrieved from Publieke Dienstverlening op de Kaart (PDOK), a geo-information service of the Dutch government (PDOK, 2016). The data belongs to the Actueel Hoogtebestand Nederland 3 (AHN3) dataset, which is being scanned from 2014 to 2019 and

currently covers about 45% of the country (AHN, 2016; PDOK, 2016). The dataset has a point density of 6 to 10 points per square meter and has information on the height with multiple discrete return values and includes intensity data. The dataset is retrieved in the first quarter of each year (our study area was scanned in 2015) and consequently the deciduous vegetation was leafless during this time (AHN, 2016). While this decreases the reflectance of vegetation, a visual inspection showed the return signal is still strong enough to retrieve a good scan of the vegetation.

4 Method

The method is summarized in figure 3. It consists of three main phases: (i) feature extraction, (ii) classification, and (iii) delineation of linear elements.

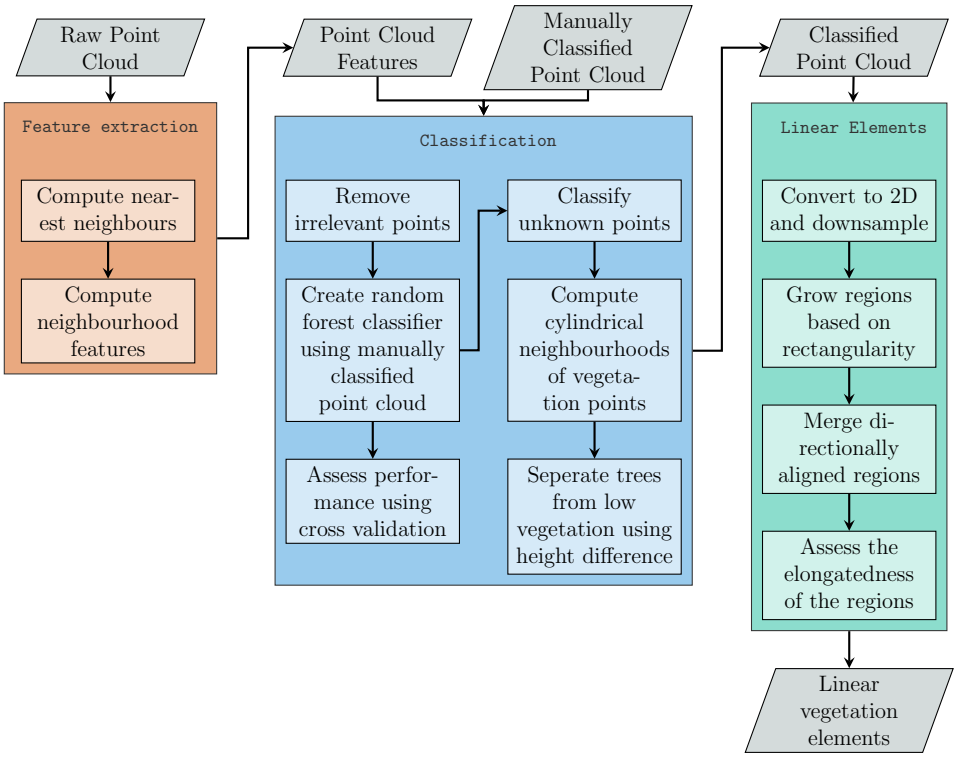

Figure 3: Overview of the method in a work flow diagram.

4.1 Feature extraction

A point cloud $\mathcal{P}$ is a set of points $\{p_1, p_2, \ldots, p_n\} \in \mathbb{R}^3$. Each point p_i consists of the X, Y and Z coordinates of the point and additional information on the intensity (I) of the return signal, the return number, and the number of returns (R_t) at p_i is known (figure 1). A normalized return number (R_n) was calculated by dividing the return number by the number of returns (Guo et al., 2011).

For each point p_i a neighbourhood set $\mathcal{N}_i$ of points $\{q_1, q_2, \ldots, q_k\}$ is defined, where $q_1 = p_i$. This neighbourhood was determined using a k-nearest neighbours method with a k of 50. This method was used instead of spherical neighbourhood of a certain radius, because it is harder to define a suitable radius which works for fluctuating point densities within and across datasets (Weinmann et al., 2014).

To determine these neighbours a k-d tree data structure was constructed, which is a multidimensional binary search tree that can be used for efficient nearest neighbour searches (Bentley, 1975).

For each neighbourhood some basic geometric properties are computed (Weinmann et al., 2015):

Height difference:
$$\Delta_{Z_i} = \max_{j:\mathcal{N}_i}(q_{Z_j}) - \min_{j:\mathcal{N}_i}(q_{Z_j}) \tag{1}$$

Height standard deviation:
$$\sigma_{Z_i} = \sqrt{\frac{1}{k}\sum_{j=1}^{k}(q_{Z_j} - \overline{q_Z})^2} \tag{2}$$

Local radius:
$$r_{l_i} = \max_{j:\mathcal{N}_i}(|p_i - q_j|) \tag{3}$$

Local point density:
$$D_i = \frac{k}{\frac{4}{3}\pi r_{l_i}^3} \tag{4}$$

Additionally the neighbourhood of a point can be used to characterize local surface features as first described by Hoppe et al. (1992), and further developed by Pauly et al. (2002), who used the local structure tensor to estimate the surface normals (the vector perpendicular to the surface ($\vec{N} = (N_x, N_y, N_z)$)) and to define the surface variation (equation 12). The structure tensor describes the directions of the neighbourhood of a point by determining the covariance matrix of the X, Y and Z coordinates of the points and computing the eigenvalues ($\lambda_1, \lambda_2, \lambda_3$, where $\lambda_1 > \lambda_2 > \lambda_3$) and eigenvectors of this matrix. The magnitude of the eigenvalues of this covariance matrix describe the spread of points in the direction of the eigenvector. By ranking the eigenvectors based on the respective eigenvalues the spread in the three principle directions of a local point cloud can be quantified. This

was expanded upon by West et al. (2004) to the following eight structure tensor features:

$$\text{Linearity:} \qquad L_\lambda = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{5}$$

$$\text{Planarity:} \qquad P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{6}$$

$$\text{Sphericity:} \qquad S_\lambda = \frac{\lambda_3}{\lambda_1} \tag{7}$$

$$\text{Omnivariance:} \qquad O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3} \tag{8}$$

$$\text{Anisotropy:} \qquad A_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1} \tag{9}$$

$$\text{Eigenentropy:} \qquad E_\lambda = -\lambda_1 \ln(\lambda_1) - \lambda_2 \ln(\lambda_2) - \lambda_3 \ln(\lambda_3) \tag{10}$$

$$\text{Sum of eigenvalues:} \qquad \sum_\lambda = \lambda_1 + \lambda_2 + \lambda_3 \tag{11}$$

$$\text{Local surface variation:} \qquad C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \tag{12}$$

$$\tag{13}$$

To avoid using an unnecessarily large feature set the correlation between all the features was checked by calculating pearson coefficients. Of the pairs of two extremely correlated features (above 0.98), one was removed. This resulted in the removal of anisotropy, which is negatively correlated with sphericity, and eigenentropy, which is correlated with omnivariance.

Consequently the resulting feature set used for the classification was $\{I, R_t, R_n, \Delta_Z, \sigma_Z, r_l, D, N_z, L_\lambda, P_\lambda, S_\lambda, O_\lambda, \sum_\lambda, C_\lambda\}$.

4.2 Classification

Since the linear vegetation elements consist of either trees or low vegetation (shrubs and small trees) the point cloud needs to be classified into *trees*, *low vegetation*, and *irrelevant* points. The difference between trees and low vegetation is ambiguous and is dependent on interpretation. In previous research this boundary has been set at varying height values, ranging from 1.5 (Bork and Su, 2007), to 3.5 (Antonarakis et al., 2008), to 4 (Koukoulas and Blackburn, 2005), to 5 meters (Khosravipour et al., 2014). We set this boundary at 4m. The irrelevant points are returns of which the linearity does not need to be checked. This may include very low vegetation (herbs), bare ground, water, buildings, and other man made objects.

4.2.1 Data trimming

To speed up the classification first points were removed which were certain not relevant. This was done based on the planarity (equation 6) and sphericity (equation 7) features discussed earlier. Since scanning vegetation (which is larger than herbs) always results in a locally very scattered point cloud, with points spread out throughout the 3D space, points with a locally planar neighbourhood can be removed. That is to say points with a high planarity value, $P_\lambda > 0.7$, and a low sphericity value, $S_\lambda < 0.05$, were removed. These values were manually selected in a conservative way to make sure every point removed was certainly not relevant, while still removing a large portion of points.

4.2.2 Supervised classification

The remaining point cloud was classified into *vegetation* and *irrelevant* using a random forest classifier. The random forest algorithm creates a collection of decision trees each based on a random subset of the training data (Ho, 1998). These decision trees are computed by a Classification And Regression Tree (CART) algorithm, which creates splits that minimize a gini impurity index (Breiman et al., 1984). This impurity index is the probability a randomly picked sample would be misclassified, given it was randomly classified conform the distribution of classes. For each point, each tree in the forest determines a class, and the class which gets selected by the majority of the trees is chosen as the final classification (Breiman, 2001).

Training and testing data was created by analysing the research area and manually segmenting areas of *vegetation* and *irrelevant*. This was done based on the point cloud and aerial photos (PDOK, 2015).

Because the point cloud is trimmed based on planarity/sphericity and it concerns an agricultural landscape it has become very imbalanced, having a lot more *vegetation* than *irrelevant* points, which was also very noticeable in the training and testing data. Imbalanced training data can lead to undesirable classification results (He and Garcia, 2009). Therefore a balanced random forest was used, where instead of the decision trees being made using bootstrap samples of the entire training dataset, a bootstrap sample was taken from only the minority class and a random sample was taken of the majority class based on the size of the minority class sample (Chen et al., 2004). Employing enough trees eventually all majority class data gets used, while still maintaining a balance between the two classes. The size of the majority sample compared to the minority sample can be adjusted to find the best balance.

The random forest parameters (i.e. maximum depth, maximum number of features, minimal samples per leaf, minimal samples per split) and the ratio between minority and majority samples were optimized using a cross validated grid search.

4.2.3 Accuracy Assessment

To assess the accuracy of the classification a confusion matrix was used. Such a matrix shows the predicted and the actual classes of the tested pixels/points. It gives an overview of the performance of the classification in that it identifies and quantifies the errors. Using this matrix a precision (user's accuracy), recall (producer's accuracy), and overall accuracy can be calculated (Stehman, 1997), but these metrics are unable to provide a good picture of the performance of a classifier when the data is very imbalanced.

Instead, three metrics, which provide a useful indication of performance even when dealing with a very imbalanced dataset, were used (Sun et al., 2009; López et al., 2013). These are: (i) the receiver operating characteristic (ROC) curve (Bradley, 1997), (ii) the Matthew's correlation coefficient (MCC) (Matthews, 1975), and (iii) the geometric mean (Kubat et al., 1998).

To create a ROC-curve the true positive rate is plotted against the false positive rate at various decision thresholds. The area under a ROC-curve (AUC) is a measure for the performance of the classifier (Bradley, 1997).

The MCC analyses the correlation between the observed and the predicted data and is defined as follows:

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{14}$$

where TP are the true positives, TN the true negatives, FP the false positives, and FN the false negatives. It often gives a more balanced evaluation of a classifier than the precision, recall and accuracy (Baldi et al., 2000) and it is an effective metric for unbalanced datasets (Kohavi et al., 1995).

The geometric mean of the recall of both classes evaluates the balanced performance of the classifier for the two classes (Kubat et al., 1998; Sun et al., 2009).

These scores were acquired by performing a 10-fold cross validation. This is done by splitting the data into 10 randomly mutually exclusive subsets and using a subset as testing data on a classifier trained on the remaining data (Kohavi et al., 1995). This method is effective against overfitting and does not further reduce the training data (which a test and validation set would).

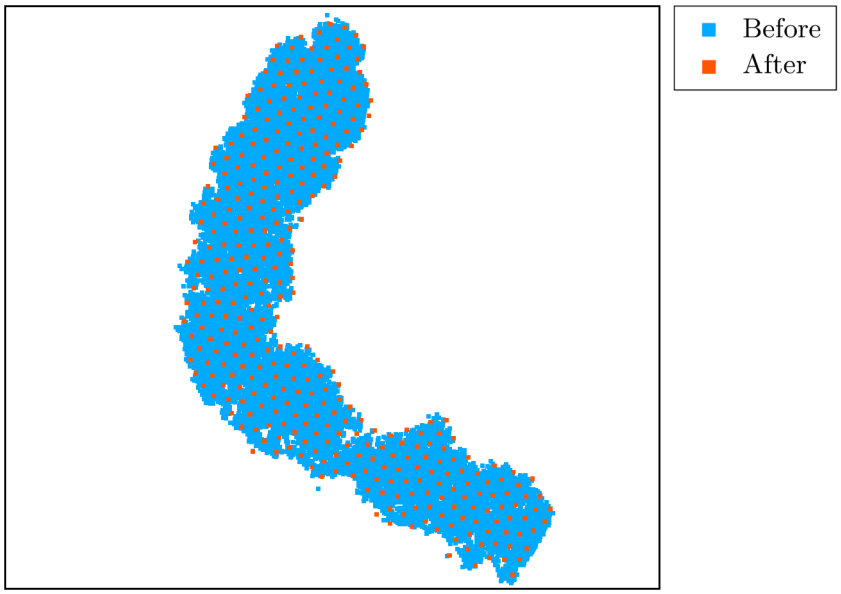

Figure 4: An example showing the tree points before and after downsampling.

To assess which features are most influential when separating vegetation from non-vegetation a feature importance analysis was performed. This is done by summing the decreases in gini impurity due to a feature in all the trees divided by the total amount of trees (Breiman, 2002).

4.2.4 Delineating trees and low vegetation

To divide the vegetation class into *tree* and *low vegetation* for each point a cylindrical neighbourhood was determined, which includes all points within a cylinder with infinite height and a radius of 2 meters. Within this neighbourhood the height difference (equation 1) was computed. As mentioned earlier, if this value exceeded 4 meters the point was classified as a *tree* point, otherwise as a *low vegetation* point.

4.3 Delineating Linear Elements

To delineate linear from non-linear vegetation elements the two classes were segmented into rectangular objects, since those can be assessed for linearity by their length and width. This is done by growing regions within the relevant points as long as the shape remains rectangular.

Since the linearity of an object is solely determined by the 2D spatial distribution of points, the point cloud was converted to 2D by removing the

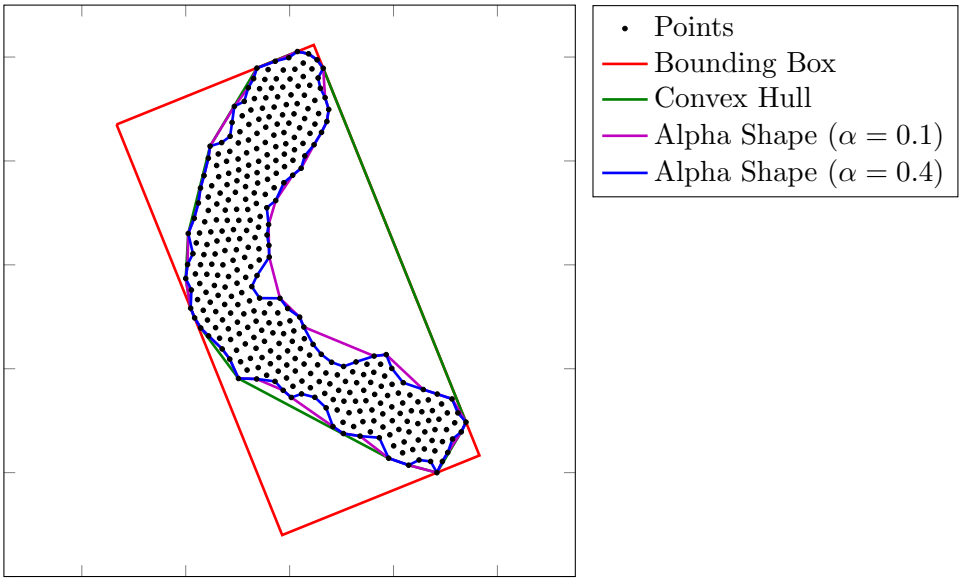

Figure 5: An example showing the different hulls used for determining the rectangularity.

Z-coordinate from all the points. To speed up the computation the 2D point cloud was spatially downsampled to 1 meter distance between all points for low vegetation and 2 meter for trees (figure 4). Low vegetation objects are often smaller than trees and consequently require more points for an accurate delineation. This leads to a slightly less precise delineation, but results in a substantially decreased computation time.

4.3.1 Rectangularity

The rectangularity of an object can be described as the ratio between the area of an object and the area of its minimum bounding rectangle (Rosin, 1999).

The minimum bounding rectangle (figure 5) is computed with rotating calipers as described by Toussaint (1983). First a convex hull is determined for the points, which is the smallest convex set of points which contains every point (figure 5), by using the QuickHull algorithm as defined by Preparata and Shamos (1985). The minimum bounding rectangle has a side collinear with one of the edges of the convex hull (Freeman and Shapira, 1975). Consequently while rotating by the angles of the edges rectangles can be created using the minima and maxima of the coordinates in the rotated system. Keeping track of the rectangle with the minimal area the minimum bounding

rectangle can be found.

The area of the object can be found by computing an alpha shape of the set of points belonging to the object (Edelsbrunner et al., 1983). An alpha shape is a hull, similar to a convex hull, which describes the shape of a set of points (figure 5). It is created by computing a Delaunay triangulation of the points (Delaunay, 1934) and removing the triangles with a circumradius higher than $1/\alpha$, where α is a parameter which consequently influences the amount of triangles removed from the triangulation and thus the shape and area of the alpha shape. Higher alphas lead to more complex shapes, while lower ones to more smooth shapes (figure 5).

4.3.2 Region Growing

To segment the point cloud into rectangular objects a region growing segmentation technique was used. This method was introduced by Besl and Jain (1988) and has since been an effective approach to segment a point cloud (Tóvári and Pfeifer, 2005; Rabbani et al., 2006; Nurunnabi et al., 2012; Vosselman, 2013; Elberink and Kemboi, 2014; Vo et al., 2015). The region growing method consists of two main steps: seed selection and region growing. The seed locations can be selected randomly or based on some properties of the points. The growing is based on a similarity criterion based on the proximity and attributes of the points.

The region growing used in this method is a bit different. Instead of using the attributes to compare the points and grow based on some similarity criteria, the regions are grown based only on proximity and a constraint of rectangularity (figure 6). The seed selection is done based on the coordinates to minimize the chances of starting at boundary regions.

Thus the point with the minimal x-coordinate and its 20 closest neighbours were used as the starting region (given this region is rectangular, otherwise the point is discarded and the process repeated) and subsequently points were added as long as the region's rectangularity did not drop below a threshold of 0.55. This threshold was determined by experimentation and manual evaluation. Once no more new points could be added to the region the procedure was repeated for the next region until all the points are checked.

4.3.3 Merging Objects

The objects can still be quite fragmented after they have been grown, especially for curved regions. To fix this, objects were merged which were (i) in proximity of each other, (ii) aligned with each other, and (iii) facing about

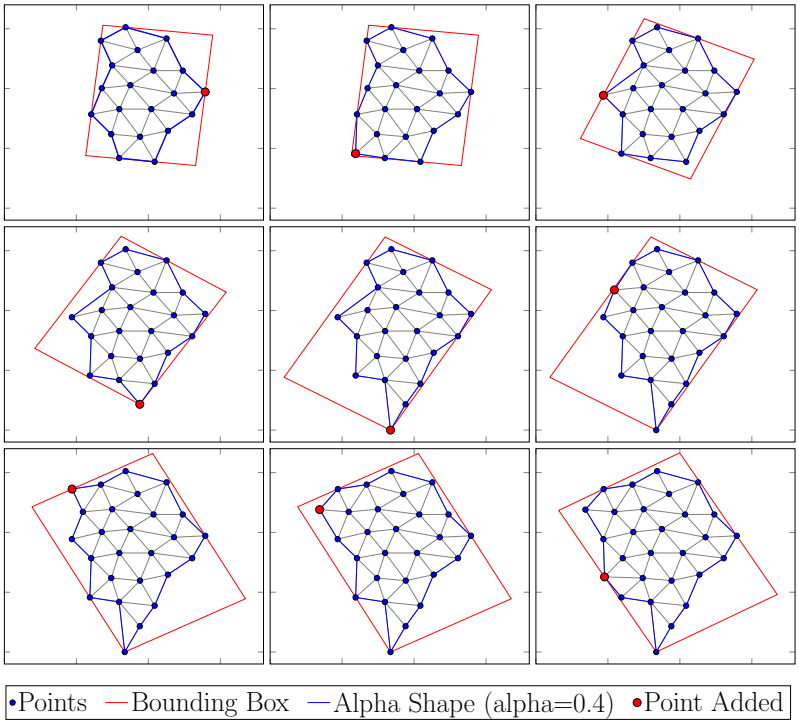

Figure 6: An example showing the region growing process.

the same direction. The direction was determined by computing the angle between one of the long sides of the minimum bounding box and the x-axis. The alignment was checked by comparing the angle of the line between the two centre points of the objects with the directions. Once merged the lengths of the objects were added and the maximum of the widths taken as the new width.

4.3.4 Elongatedness

The resulting rectangular regions were assessed for linearity by determining the elongatedness of an object, which is defined as the ratio between its length and its width (Nagao and Matsuyama, 2013).

The minimum elongatedness of objects was set at 2.5, based on a manual trial and error. To prevent long, but wide patches of forest to be classified as linear elements a maximum width of 60 meters was used.

4.3.5 Accuracy Assesment

To assess the accuracy of the delineation of linear objects a manual delineation was done. First a polygon was made both from the tree and low

vegetation points using alpha shapes. Subsequently the two polygons were manually further segmented into objects. Finally the polygons belonging to a linear object according to our interpretation were manually marked as being linear. The resulting map was compared with the automated delineation. This was done by a difference map and confusion matrices where we compared the true positive, true negative, false positive and false negative in area. Since this data is, unlike before, not very imbalanced we can use traditional metrics for accuracy. Therefore a user's, producer's and overall accuracy was calculated, as well as an F1, kappa and MCC score (Congalton and Green, 2008).

4.4 Software & Hardware

We chose to use solely free and open source software (FOSS), for two main reasons: (i) to avoid 'black boxes', where it is unclear exactly what calculations or algorithms are being used to produce the results, and (ii) to make the method more easily runnable with cloud- or supercomputing solutions, and thus able to compute for very large areas.

The large majority was made in Python (2.7.12) with the NumPy (1.11.2), SciPy (0.18.1), pandas (0.18.1), scikit-learn (0.17.1) and Shapely (1.5.13) libraries. For the preprocessing of data LASTools (version 160429) and CloudCompare (v2.8) were used and Cloudcompare was also used for visualizing and downsampling the point cloud.

For the computation a desktop computer was used with an Intel Xeon E3-1220 3.1 GHz quad-core processor and 16GB of RAM.

5 Results

5.1 Classification

The intermediate result after the classification of vegetation is shown in figure 7. The accuracy of this classification is presented using a confusion matrix (table 1) and the measures for unbalanced datasets in table 2.

The ROC-AUC of 0.97 shows the two classes can be separated well, supported by the Matthews correlation coefficient of 0.60, indicating a positive correlation between the predicted and observed classes, and the geometric mean of 0.84. The confusion matrix shows a recall of 0.98 for vegetation and of 0.72 for non-vegetation.

The importance of the different features is shown in figure 8. It is clear the number of returns is the most important feature when filtering out vegetation.

This is not surprising as vegetation is the most prevalent object scanned which produces multiple returns. Not every object with multiple returns is vegetation though (the train power line for example has multiple returns), and not every point with a single return is non-vegetation (especially the edges of low vegetation quite often only have one return). The other features help to further determine to which class a point belongs.

The classification took 1 hour, 6 minutes, and 36 seconds to compute, of which 11m 37s were used for training the classifier, 37m 36s to classify vegetation, and 17m 23s to classify trees and low vegetation. The computation of the features took 1h 25m 34s, of which 17m 34s were used for computing the nearest neighbours and 68m to compute the features.

5.2 Linear Elements

To delineate the linear elements first the vegetation points were segmented into rectangular-like objects of which the results are presented in figure 9.

The extracted linear objects are shown in figure 10, as well as the result of the manual delineation and the differences between the two. The correspond-

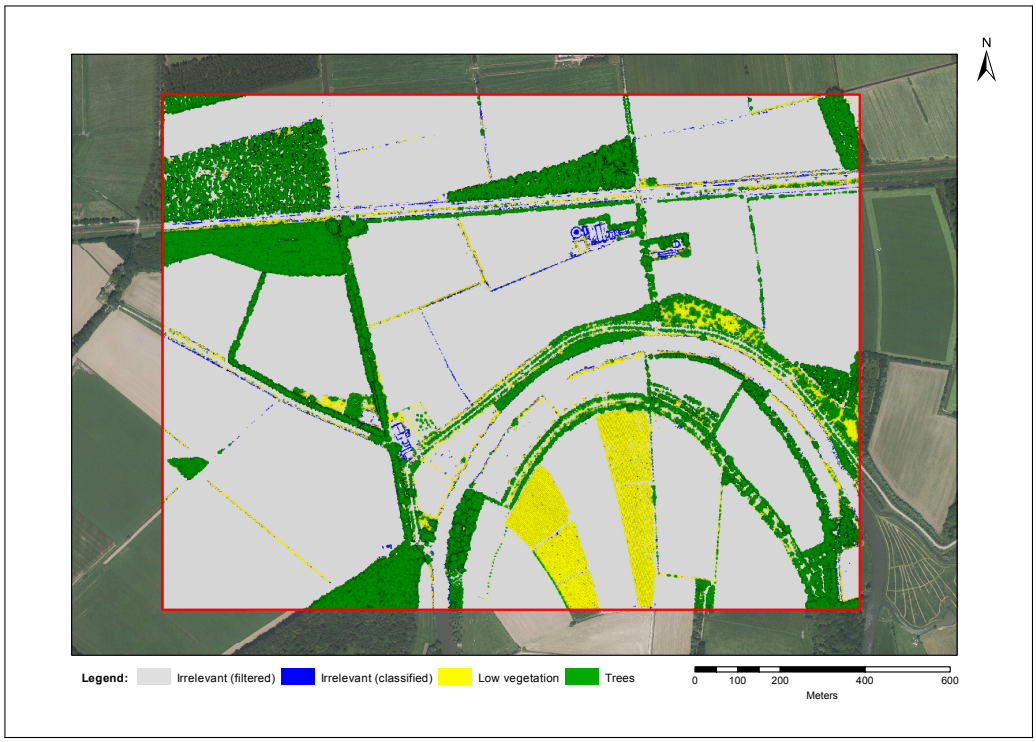

Figure 7: A map of the classification results.

Table 1: A confusion matrix showing the predicted classes against the actual classes of the 10 fold cross-validation accumulated.

Predicted	Vegetation	Irrelevant	Recall
Actual			
Vegetation	1159762	23438	0.98
Irrelevant	6416	16706	0.72

Table 2: Assessment of the accuracy using the average ROC-AUC, MCC and geometric mean of the 10 fold cross-validation.

Metric	Score
ROC-AUC	0.97
MCC	0.60
Geometric Mean	0.84

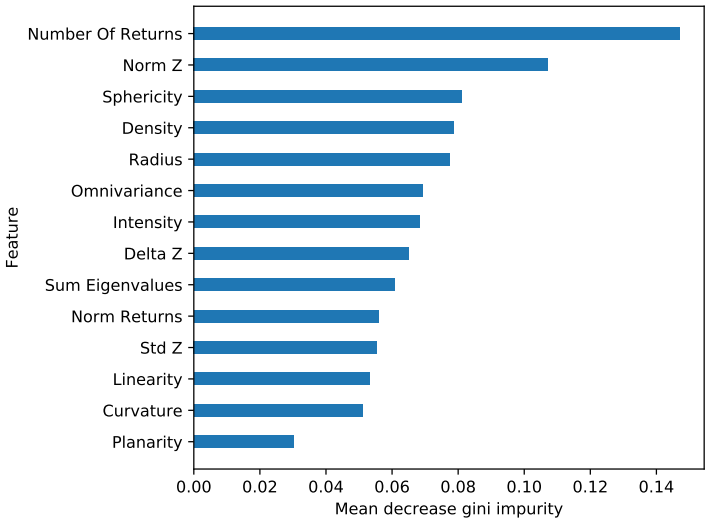

Figure 8: The importance of each feature in the classifier, based on the mean decrease in impurity by the feature. Higher is more important.

ing confusion matrices are presented in table 3 and 4, with the associated scores shown in table 5.

The scores are similar for both classes with an overall accuracy of 0.87 for trees and 0.90 for low vegetation, f1 of 0.77 and 0.76, kappa of 0.70 and 0.67, and MCC of 0.70 and 0.67. Although there are errors, the majority of linear elements gets successfully delineated. Especially the areas not belonging to a linear element get separated well, with user's and producer's accuracies of 0.90 and 0.91 for trees, and 0.93 and 0.94 for low vegetation, respectively. The delineation of linear elements is lower at 0.77 and 0.75 for trees, and 0.78 and 0.76 for low vegetation.

The automated process took 4 hours, 47 minutes and 7 seconds to compute for low vegetation, and 1h, 47m and 14s for trees. Since the process was running for trees and low vegetation simultaneously on separate CPU cores the total process took 4h, 47m and 7s.

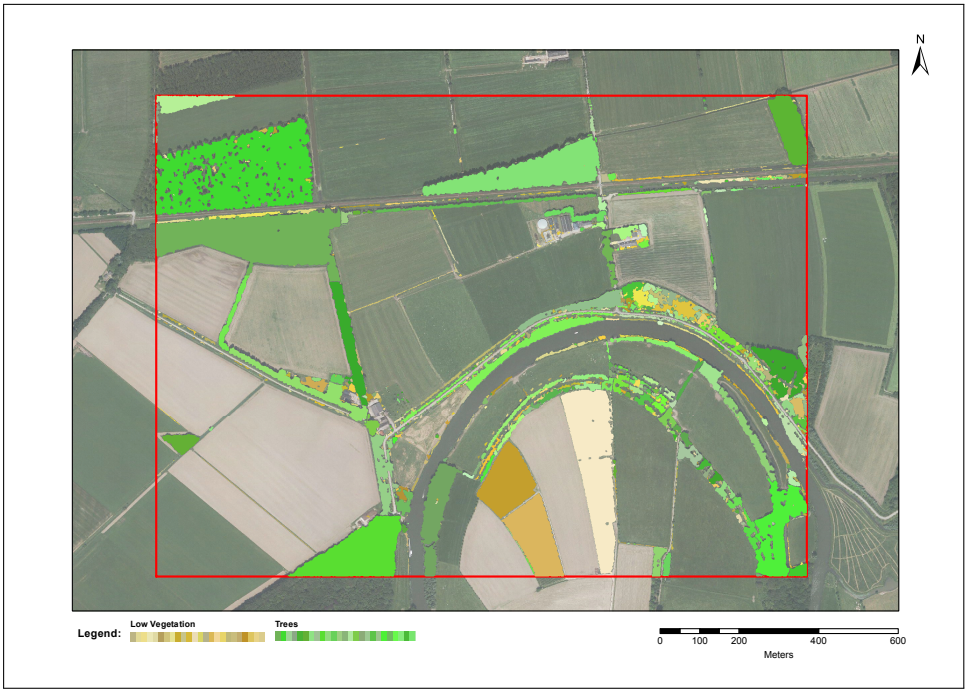

Figure 9: The resulting rectangular objects.

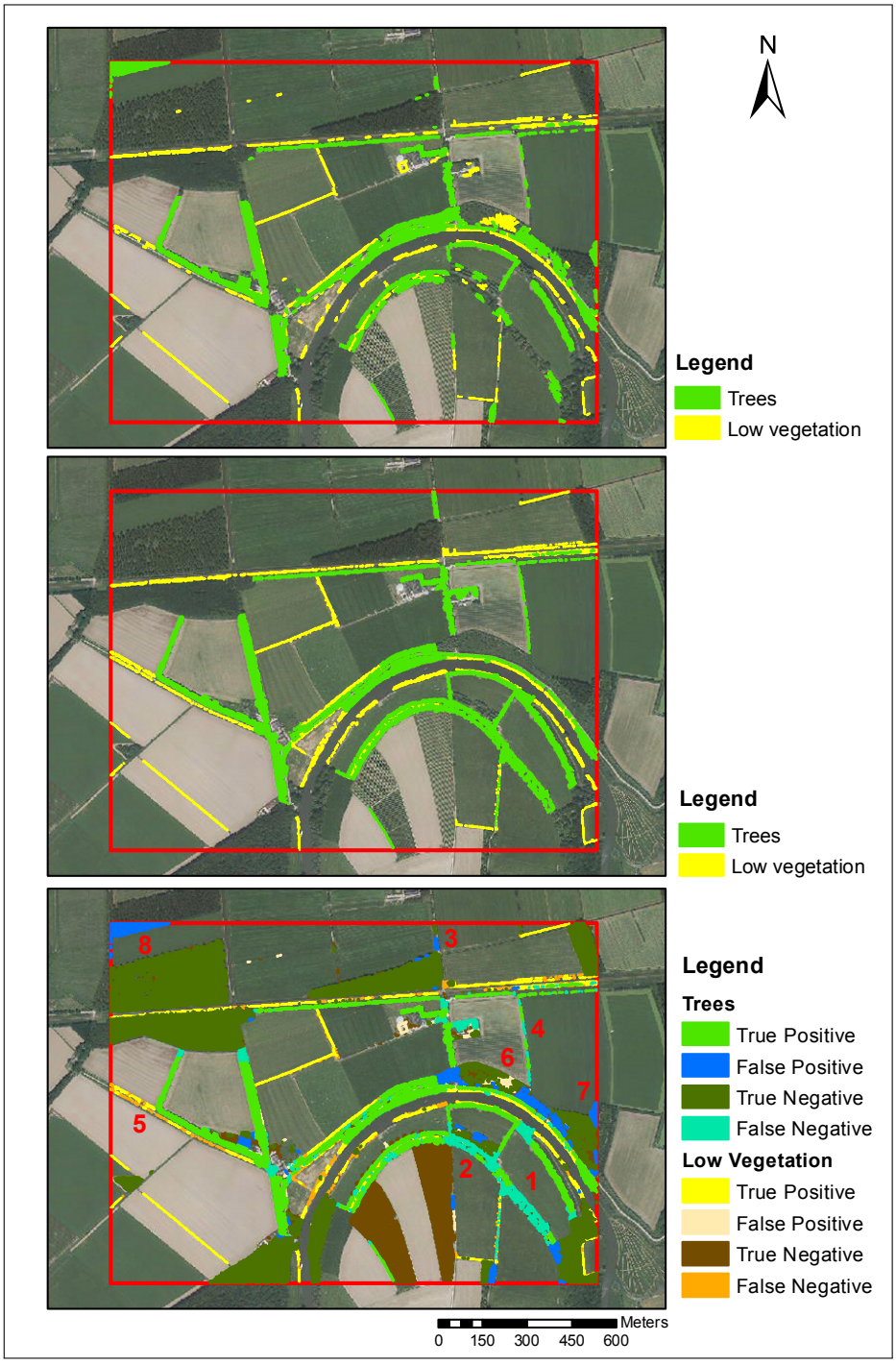

Figure 10: The result of the delineation of linear objects. Top: automated delineation, middle: manual delineation, bottom: difference between the two.

Table 3: A confusion matrix showing the automated against the manual delineation for trees in square meters. Overall accuracy shown in the bottom right.

Automated	Linear	Non-linear	Producer's accuracy
Manual			
Linear	70206	23188	0.75
Non-linear	20948	226727	0.91
User's accuracy	0.77	0.90	0.87

Table 4: A confusion matrix showing the automated against the manual delineation for low vegetation in square meters. Overall accuracy shown in the bottom right.

Automated	Linear	Non-linear	Producer's accuracy
Manual			
Linear	19104	6094	0.76
Non-linear	5243	83506	0.94
User's accuracy	0.78	0.93	0.90

Table 5: Assessment of the accuracy using the f1, kappa and MCC scores.

Metric	Low vegetation	Trees
f1	0.76	0.77
Kappa	0.67	0.70
MCC	0.67	0.70

6 Discussion

6.1 Classification

The classification results are fairly accurate, looking at the accuracy assessment metrics. Especially the vegetation class is classified very well, having 98% of the points classified as vegetation also being vegetation. The irrelevant class, which is mostly composed of buildings, railway power lines and ditches, is less accurate with 72% of the points correctly classified.

When analysing these statistics it is important to take the removal of the majority of the points in consideration. A major part of the irrelevant points were removed before the classification. The irrelevant points left share many similarities with vegetation points and are consequently much harder to classify correctly.

Improving the classification could be tried in a number of ways. We identified four areas which can be changed for a better result.

Firstly, the result of the feature importance assessment could be used to make a selection of the more important features and only use those in the classification. This could be a way to improve the result, as irrelevant features can decrease the performance of a classifier, and decrease the computation time, as less features lead to simpler trees (Rogers and Gunn, 2006).

Secondly, other techniques to tackle the unbalanced data problem could also present different results. Instead of adjusting the sampling technique within the random forest algorithm it is also possible to oversample the minority class or undersample the majority class before using a machine learning algorithm. This can be done randomly, but also in smart ways, using algorithms such as SMOTE (Synthetic Minority Over-sampling Technique) or ADASYN (Adaptive Synthetic Sampling) for oversampling (Chawla et al., 2002; He et al., 2008), and Condensed Nearest Neighbor (CNN) or Tomek Links for undersampling (Hart, 1968; Tomek, 1976).

Thirdly, the classification result might also be improved by using a different machine learning algorithm. Algorithms like SVM (Mallet et al., 2008; Zhang et al., 2013; Weinmann et al., 2015) and Adaboost (Lodha et al., 2007; Weinmann et al., 2015) have been proven to be effective classifiers and could potentially offer a better classification result.

Lastly, selecting another neighbourhood could provide better results as well. Spherical or cylindrical neighbourhoods could be used instead, or the k-nearest neighbourhood could be improved by finding optimal values for k. This can for example be done by minimizing the eigenenthropy (equation 10) for a range of k (Weinmann et al., 2014). While this results in a slightly improved classification, the computational effort required is high.

For our purpose however a complete accurate result is not necessary, as we want to delineate the linear vegetation elements for which we need most of the vegetation points correctly classified, but not all. The classification result show most of the incorrectly classified points are dispersed and surrounded by correctly classified points. These scattered points do not impact the subsequent delineation of linear objects. If desired these points could be cleaned up by looking at a neighbourhood around each point and changing its classification if a very large majority of those points are of a different class.

6.2 Delineating Linear Objects

The comparison of the manual with automated delineation shows that linear objects were generally accurately extracted, evidenced by the accuracy scores. When interpreting these scores it is important to keep in mind the degree of interpretation which is required when creating the manual classification. The rules for when a vegetation object is linear are ambiguous and thus open for varying interpretations.

The automated delineation shows a few additions and omissions with our interpretation. The linear objects at 1 and 2 (figure 10) are the biggest omission in the automated delineation compared to the manual one. Closer inspection of the rectangular objects in this area reveal that this area is segmented as a series of rectangle-like objects instead of a single more elongated element. This is the result of the distribution of the trees in rows of two to three across a dike (see figure 11). These objects are by themselves not linear and do not get merged because they are not aligned. Omissions at 3, 4 and 5 are the result of the downsampling of the point cloud before the automated delineation. The tree lines at 3 are composed of very young and thin trees and consequently have too few points for the creation of proper rectangular objects. Position 4 might suffer from this as well, but also contains physical gaps between the trees. The low vegetation at 5 has similar problems. There are also places where the automated method delineates linear objects which are not part of the manual delineation. Positions 6, 7 and 8 are examples of this. The trees at 6 are part of a larger vegetation patch which includes both low vegetation and trees. A human is prone to see this as a single patch and thus not a linear element, while a machine does not make this interpretation. The forest patches at 7 and 8 get classified as linear elements because they are on the border of the research area and are cut off. Without this cut off the objects would not be seen as a linear element.

The results show a few main limitations of the method. Firstly, the computation time is fairly high. Therefore a compromise has to be made between

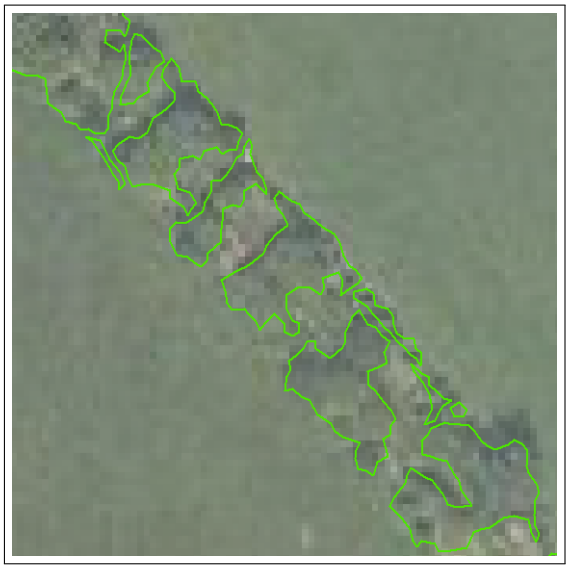

Figure 11: Detail of the omission at 1.

precision and time. The resulting downsampling of the points can cause small objects to be ignored. Secondly, the objects are not always merged correctly. The merging is done based on the orientation and alignment of the two objects, which works well in most cases. The orientation of an object however is only meaningful when it is somewhat elongated, which is not the case for single trees for example. A row of separated single trees will consequently not be merged and seen as a linear element. The alignment of objects can also prevent some objects, which could be seen as one object, to be merged. This can be seen at position 1, as the objects are in a row of unaligned pairs of trees. Lastly, when vegetation is mixed the method might delineate linear objects, while this might not be desirable. If a mixed vegetation should be seen as one object the delineation of linear objects could be done before the classification into low vegetation and trees.

Despite these limitations the method is successful in delineating the large majority of the linear elements present in the research area, which contains linear and non-linear vegetation patches of many different shapes.

6.3 Future outlook

Even though the research area was chosen to cover many different shapes of linear and non-linear vegetation patches the method should be validated using more and different areas, both within the AHN3 dataset as datasets of different countries. This should be done without retraining the classifier. If

this fails a more general applicable classifier needs to be created.

The automated nature of the method means it can be upscaled to large areas. The linear vegetation elements can be delineated for the whole AHN3 dataset for example, eventually spanning the entire country of the Netherlands. This would require quite some computational power, but high performance computing paradigms like cloud computing or supercomputing can provide the power needed. The method is written using FOSS and should be able to run in such an environment. The dataset could be split up into smaller parts and run in parallel.

National LiDAR datasets are becoming increasingly available for European countries besides the Netherlands, such as in Denmark, England, Finland, Slovenia, Spain, Sweden and Switzerland. With minor alterations the method should be able to work with these datasets as well, making it possible to cover a large part of Europe.

The resulting database of tree lines and hedges throughout the Netherlands or Europe could be used as an extra indicator for ecosystem health and biodiversity in agricultural landscapes. This could provide an improvement in ecosystem and biodiversity assessments, such as the MAES (Mapping and Assessment of Ecosystems and their Services) project (Maes et al., 2013), the SEBI (Streamlining European Biodiversity Indicators) project (Biała et al., 2012), and the high nature value farmland assessment (Paracchini et al., 2008) on a European scale and assessments of Planbureau voor de Leefomgeving (PBL) on a national level (Bouwma et al., 2014).

Additionally the results could be used for further ecological and physical geographical research. Tree lines and hedges play an important role, for example, in the distribution of species and in the microclimate of fields. They have an influence on varying fluxes, such as wind and water flows, soil erosion and animal movement (Forman and Baudry, 1984). It could consequently be used as a parameter in a species distribution model, or to analyse the relation between linear elements and certain animal species, or to estimate the stability of soil.

The method can also be used for other purposes. Other landscape elements, like roads and ditches, can also be segmented using this technique. These elements also influence the ecosystem as barriers and corridors. The automated delineation of roads is also subject of research outside of ecology (Quackenbush, 2004).

7 Conclusion

In this paper we presented a method to delineate linear vegetation elements from LiDAR point clouds. Growing regions based on the rectangularity proved a useful way to segment different objects in the vegetation which could be checked for linearity. Accuracies were high, with overall accuracies of 0.87 and 0.90, in a research area with linear and non-linear vegetation patches of different shapes and sizes. The main limitations are the balance between computation time and precision, which can be solved by using high performance computing paradigms, and the difficulty of setting good rules for merging regions. Despite these limitations the method was largely successful in its goal and is ready to be upscaled to larger areas using high performance computing paradigms like cloud computing. The ecological value of this research can then be further explored. Furthermore the method could also be used for different elongated objects, like roads, ditches and waterways.

References

AHN (2016), 'Inwinjaren AHN2 & AHN3'. [Online: accessed April 2017]. `http://www.ahn.nl/common-nlm/inwinjaren-ahn2--ahn3.html`

Antonarakis, A., Richards, K. S. and Brasington, J. (2008), 'Object-based land cover classification using airborne lidar', *Remote Sensing of Environment* **112**(6), 2988–2998.

Bailly, J., Lagacherie, P., Millier, C., Puech, C. and Kosuth, P. (2008), 'Agrarian landscapes linear features detection from lidar: application to artificial drainage networks', *International Journal of Remote Sensing* **29**(12), 3489–3508.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. and Nielsen, H. (2000), 'Assessing the accuracy of prediction algorithms for classification: an overview', *Bioinformatics* **16**(5), 412–424.

Bentley, J. L. (1975), 'Multidimensional binary search trees used for associative searching', *Communications of the ACM* **18**(9), 509–517.

Besl, P. J. and Jain, R. C. (1988), 'Segmentation through variable-order surface fitting', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10**(2), 167–192.

Biała, K., Condé, S., Delbaere, B., Jones-Walters, L. and Torre-Marín, A. (2012), Streamlining european biodiversity indicators 2020, Technical Report 11/2012, European Environment Agency.

Bork, E. W. and Su, J. G. (2007), 'Integrating lidar data and multispectral imagery for enhanced classification of rangeland vegetation: A meta analysis', *Remote Sensing of Environment* **111**(1), 11–24.

Bouwma, I., Sanders, M., op Akkerhuis, G. J., Onno Knol, J. V., de Wit, B., Wiertz, J. and van Hinsber, A. (2014), Biodiversiteit bekeken: hoe evalueert en verkent het PBL het natuurbeleid?, Technical Report 924, Planbureau voor de Leefomgeving.

Bradley, A. P. (1997), 'The use of the area under the roc curve in the evaluation of machine learning algorithms', *Pattern recognition* **30**(7), 1145–1159.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Breiman, L. (2002), 'Manual on setting up, using, and understanding random forests v3. 1', *Statistics Department University of California Berkeley, CA, USA* **1**.

Breiman, L., Friedman, J., Stone, C. J. and Olshen, R. A. (1984), *Classification and regression trees*, CRC press.

Brennan, R. and Webster, T. (2006), 'Object-oriented land cover classification of lidar-derived surfaces', *Canadian Journal of Remote Sensing* **32**(2), 162–172.

Burel, F. (1996), 'Hedgerows and their role in agricultural landscapes', *Critical reviews in plant sciences* **15**(2), 169–190.

Charaniya, A. P., Manduchi, R. and Lodha, S. K. (2004), Supervised parametric classification of aerial lidar data, *in* 'Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on', IEEE, pp. 30–30.

Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. (2002), 'Smote: synthetic minority over-sampling technique', *Journal of artificial intelligence research* **16**, 321–357.

Chehata, N., Guo, L. and Mallet, C. (2009), 'Airborne lidar feature selection for urban classification using random forests', *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **38**(3), 207–2012.

Chen, C., Liaw, A. and Breiman, L. (2004), Using random forest to learn imbalanced data, Technical Report 666, Department of Statistics, UC Berkeley.

Congalton, R. G. and Green, K. (2008), *Assessing the accuracy of remotely sensed data: principles and practices*, CRC press.

Delaunay, B. (1934), 'Sur la sphere vide', *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* **7**(793-800), 1–2.

Edelsbrunner, H., Kirkpatrick, D. and Seidel, R. (1983), 'On the shape of a set of points in the plane', *IEEE Transactions on information theory* **29**(4), 551–559.

Elberink, S. O. and Kemboi, B. (2014), 'User-assisted object detection by segment based similarity measures in mobile laser scanner data', *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **40**(3), 239.

Forlani, G., Nardinocchi, C., Scaioni, M. and Zingaretti, P. (2006), 'Complete classification of raw lidar data and 3d reconstruction of buildings', *Pattern Analysis and Applications* **8**(4), 357–374.

Forman, R. T. and Baudry, J. (1984), 'Hedgerows and hedgerow networks in landscape ecology', *Environmental management* **8**(6), 495–510.

Freeman, H. and Shapira, R. (1975), 'Determining the minimum-area encasing rectangle for an arbitrary closed curve', *Communications of the ACM* **18**(7), 409–413.

Gobster, P. H., Nassauer, J. I., Daniel, T. C. and Fry, G. (2007), 'The shared landscape: what does aesthetics have to do with ecology?', *Landscape ecology* **22**(7), 959–972.

Guo, L., Chehata, N., Mallet, C. and Boukir, S. (2011), 'Relevance of airborne lidar and multispectral image data for urban scene classification using random forests', *ISPRS Journal of Photogrammetry and Remote Sensing* **66**(1), 56–66.

Hart, P. (1968), 'The condensed nearest neighbor rule (corresp.)', *IEEE Transactions on Information Theory* **14**(3), 515–516.

He, H., Bai, Y., Garcia, E. A. and Li, S. (2008), Adasyn: Adaptive synthetic sampling approach for imbalanced learning, *in* 'Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on', IEEE, pp. 1322–1328.

He, H. and Garcia, E. A. (2009), 'Learning from imbalanced data', *IEEE Transactions on knowledge and data engineering* **21**(9), 1263–1284.

Ho, T. K. (1998), 'The random subspace method for constructing decision forests', *IEEE transactions on pattern analysis and machine intelligence* **20**(8), 832–844.

Hoppe, H., DeRose, T., Duchampt, T., McDonald, J. and Stuetzle, W. (1992), 'Surface reconstruction from unorganized points', *Computer Graphics* **26**, 2.

Im, J., Jensen, J. R. and Hodgson, M. E. (2008), 'Object-based land cover classification using high-posting-density lidar data', *GIScience & Remote Sensing* **45**(2), 209–228.

Jongman, R. (2004), 'Landscape linkages and biodiversity in european landscapes', *The new dimensions of the European landscape. Springer, Dordrecht* pp. 179–189.

Khosravipour, A., Skidmore, A. K., Isenburg, M., Wang, T. and Hussin, Y. A. (2014), 'Generating pit-free canopy height models from airborne lidar', *Photogrammetric Engineering & Remote Sensing* **80**(9), 863–872.

Kohavi, R. et al. (1995), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* 'Ijcai', Vol. 14, Stanford, CA, pp. 1137–1145.

Koukoulas, S. and Blackburn, G. A. (2005), 'Spatial relationships between tree species and gap characteristics in broad-leaved deciduous woodland', *Journal of Vegetation Science* **16**(5), 587–596.

Kubat, M., Holte, R. C. and Matwin, S. (1998), 'Machine learning for the detection of oil spills in satellite radar images', *Machine learning* **30**(2-3), 195–215.

Lefsky, M. A., Cohen, W. B., Parker, G. G. and Harding, D. J. (2002), 'Lidar remote sensing for ecosystem studies: Lidar, an emerging remote sensing technology that directly measures the three-dimensional distribution of plant canopies, can accurately estimate vegetation structural attributes and should be of particular interest to forest, landscape, and global ecologists', *AIBS Bulletin* **52**(1), 19–30.

Lim, K., Treitz, P., Wulder, M., St-Onge, B. and Flood, M. (2003), 'Lidar remote sensing of forest structure', *Progress in physical geography* **27**(1), 88–106.

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

López, V., Fernández, A., García, S., Palade, V. and Herrera, F. (2013), 'An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics', *Information Sciences* **250**, 113–141.

Maas, H.-G. (1999), The potential of height texture measures for the segmentation of airborne laserscanner data, *in* 'Fourth international airborne remote sensing conference and exhibition/21st Canadian symposium on remote sensing', pp. 154–161.

Maes, J., Teller, A., Erhard, M., Liquete, C., Braat, L., Berry, P., Egoh, B., Puydarrieux, P., Fiorina, C., Santos, F. et al. (2013), Mapping and assessment of ecosystems and their services, Technical Report EUR 27143 EN, Joint Research Center - Institute for Environment and Sustainability.

Mallet, C., Bretar, F. and Soergel, U. (2008), 'Analysis of full-waveform lidar data for classification of urban areas', *Photogrammetrie Fernerkundung Geoinformation* **5**, 337–349.

Matthews, B. W. (1975), 'Comparison of the predicted and observed secondary structure of t4 phage lysozyme', *Biochimica et Biophysica Acta (BBA)-Protein Structure* **405**(2), 442–451.

Meyer, B. C., Wolf, T. and Grabaum, R. (2012), 'A multifunctional assessment method for compromise optimisation of linear landscape elements', *Ecological Indicators* **22**, 53–63.

Nagao, M. and Matsuyama, T. (2013), *A structural analysis of complex aerial photographs*, Springer Science & Business Media.

Niemeyer, J., Rottensteiner, F. and Soergel, U. (2012), 'Conditional random fields for lidar point cloud classification in complex urban areas', *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **1**(3), 263–268.

Niemeyer, J., Rottensteiner, F. and Soergel, U. (2014), 'Contextual classification of lidar data and building object detection in urban areas', *ISPRS journal of photogrammetry and remote sensing* **87**, 152–165.

Niemeyer, J., Wegner, J. D., Mallet, C., Rottensteiner, F. and Soergel, U. (2011), Conditional random fields for urban scene classification with full waveform lidar data, *in* 'Photogrammetric Image Analysis', Springer, pp. 233–244.

Nurunnabi, A., Belton, D. and West, G. (2012), Robust segmentation in laser scanning 3d point cloud data, *in* 'Digital Image Computing Techniques and Applications (DICTA), 2012 International Conference on', IEEE, pp. 1–8.

Paracchini, M. L., Petersen, J.-E., Hoogeveen, Y., Bamps, C., Burfield, I. and van Swaay, C. (2008), High nature value farmland in europe, Technical Report EUR 23480 EN, Joint Research Center - Institute for Environment and Sustainability.

Pauly, M., Gross, M. and Kobbelt, L. P. (2002), Efficient simplification of point-sampled surfaces, *in* 'Proceedings of the conference on Visualization'02', IEEE Computer Society, pp. 163–170.

PDOK (2015), 'Luchtfoto (PDOK-achtergrond)'. [Online: accessed March 2017].
`https://www.pdok.nl/nl/service/wms-luchtfoto-pdok-achtergrond`

PDOK (2016), 'AHN3 downloads'. [Online: accessed March 2017].
`https://www.pdok.nl/nl/ahn3-downloads`

Preparata, F. P. and Shamos, M. (1985), *Computational geometry: an introduction*, Springer Science & Business Media.

Quackenbush, L. J. (2004), 'A review of techniques for extracting linear features from imagery', *Photogrammetric Engineering & Remote Sensing* **70**(12), 1383–1392.

Rabbani, T., Van Den Heuvel, F. and Vosselmann, G. (2006), 'Segmentation of point clouds using smoothness constraint', *International archives of photogrammetry, remote sensing and spatial information sciences* **36**(5), 248–253.

Rogers, J. and Gunn, S. (2006), Identifying feature relevance using a random forest, *in* 'Subspace, Latent Structure and Feature Selection', Springer, pp. 173–184.

Rosin, P. L. (1999), 'Measuring rectangularity', *Machine Vision and Applications* **11**(4), 191–196.

SDFE (2017), 'DHM/Punktsky'. [Online: accessed June 2017].
`https://download.kortforsyningen.dk/content/dhmpunktsky`

Song, J.-H., Han, S.-H., Yu, K. and Kim, Y.-I. (2002), 'Assessing the possibility of land-cover classification using lidar intensity data', *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* **34**(3/B), 259–262.

Spellerberg, I. F. and Sawyer, J. W. (1999), *An introduction to applied biogeography*, Cambridge University Press.

Stehman, S. V. (1997), 'Selecting and interpreting measures of thematic classification accuracy', *Remote sensing of Environment* **62**(1), 77–89.

Stoate, C., Boatman, N., Borralho, R., Carvalho, C. R., De Snoo, G. and Eden, P. (2001), 'Ecological impacts of arable intensification in europe', *Journal of environmental management* **63**(4), 337–365.

Sun, Y., Wong, A. K. and Kamel, M. S. (2009), 'Classification of imbalanced data: A review', *International Journal of Pattern Recognition and Artificial Intelligence* **23**(04), 687–719.

Tomek, I. (1976), 'Two modifications of cnn', *IEEE Trans. Systems, Man and Cybernetics* **6**, 769–772.

Toussaint, G. T. (1983), Solving geometric problems with the rotating calipers, *in* 'Proc. IEEE Melecon', Vol. 83, p. A10.

Tóvári, D. and Pfeifer, N. (2005), 'Segmentation based robust interpolation-a new approach to laser data filtering', *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **36**(3/19), 79–84.

Turner, M. G. (1989), 'Landscape ecology: the effect of pattern on process', *Annual review of ecology and systematics* pp. 171–197.

Van der Zanden, E. H., Verburg, P. H. and Mücher, C. A. (2013), 'Modelling the spatial distribution of linear landscape elements in europe', *Ecological indicators* **27**, 125–136.

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

Vosselman, G. (2013), 'Point cloud segmentation for urban scene classification', *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **1**(2), 257–262.

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

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

West, K. F., Webb, B. N., Lersch, J. R., Pothier, S., Triscari, J. M. and Iverson, A. E. (2004), Context-driven automated target detection in 3d data, *in* 'Defense and Security', International Society for Optics and Photonics, pp. 133–143.

Yan, W. Y., Shaker, A. and El-Ashmawy, N. (2015), 'Urban land cover classification using airborne lidar data: A review', *Remote Sensing of Environment* **158**, 295–310.

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

8 Appendices

8.1 Code

The python code with documentation is available on GitHub at:
`https://github.com/clucas111/delineating-linear-elements`

8.2 Data

The data is available at:
`https://drive.google.com/open?id=0BwxnnB9jCFE5Q2dVRlRxUHBlV1E`

8.3 GIS

The maps, rasters, and features are available on GitHub at:
`https://github.com/clucas111/delineating-linear-elements`

8.4 Ground survey

The photos are available at:
`https://drive.google.com/open?id=0BwxnnB9jCFE5cmJIUm1hbzhrb0k`
The approximate locations of the photos are shown in figure 12. A more detailed map is available in the folder on Google Drive.

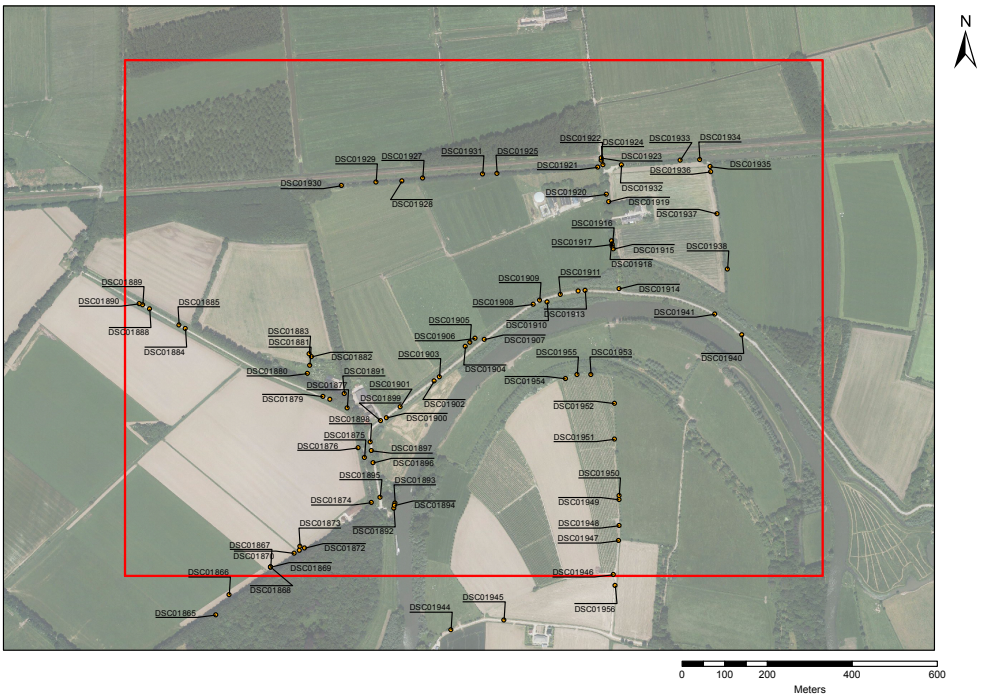

Figure 12: The approximate locations of the photos.

8.5 Detailed Work Flow

8.5.1 Data preprocessing

- Download LAZ data from PDOK
- Extract study area to LAS using laszip (LASTools)
- Spatially downsample to 0.3m distance between all points using CloudCompare (reduces effects of overlapping scans on computation of features)
- Convert LAS to CSV using las2txt (LASTools)
- Read CSV into python using pandas

8.5.2 Feature extraction

- Compute nearest neighbours using a k-d tree
- Compute basic geometric properties and structure tensors of neighbourhoods

- Compute structure tensor features
- Remove irrelevent points based on sphericity and planarity
- Compute normalized return number
- Write point features to CSV

8.5.3 Classification

- Data preparation
  - Manually segment areas of the two classes in the point cloud using CloudCompare's segment tool
  - Save the two segmented point clouds as CSV
  - Read the two CSV files into python as dataframes using pandas
  - Merge the two dataframes, adding a column indicating the class
  - Define the feature space to use during classification
  - Drop highly correlated features from the feature space by computing pearson correlation coefficient between all features.
- Parameter optimization (Grid Search)
  - Define a parameter grid to search in
  - Split the data into 3 folds
  - Loop through each fold and through each combination in the parameter grid
    - Create a balanced random forest classifier
    - Fit the training data to classifier
    - Assess the performance of classifier using the accuracy metrics
  - Choose the best parameters
- Validate classifier (Cross validation)
  - Split the data into 10 folds
  - Loop through each fold
    - Create a balanced random forest classifier with optimized parameters
    - Fit the training data to classifier

        * Assess the performance of classifier using the accuracy metrics

- Classify vegetation
  - Load CSV of the entire point cloud with computed features
  - Create a balanced random forest classifier with optimized parameters
  - Fit classifier with manually segmented data
  - Predict classes of the unknown points
- Classify trees & vegetation
  - Compute cylindrical neighbourhoods using a k-d tree
  - Classify into trees & vegetation based on height difference
- Write x, y, z coordinates and class of points to CSV

8.5.4 Delineate linear elements

- Data preparation
  - Load CSV with the points and their classes
  - Write CSV with just x and y coordinates for each class
  - Downsample point cloud using the CSVs and CloudCompare
  - Load resulting CSVs into python
  - Shift coordinates to 0, 0
- Delineate linear elements - Manual
  - Create polygon from all points from each class
  - Export polygons to shapefiles
  - Load shapefiles in ArcGIS
  - Single to multi polygon (ArcGIS tool)
  - Manually delineate linear elements in resulting polygons
- Delineate linear elements - Automated
  - Segment rectangular regions
    * While there are points left:

        - Initiate starting region (point with minimal x and its 19 closest neighbours)
        - Add points as long as region stays above rectangularity threshold
    - Convert regions to polygons
    - Merge objects if they are in proximity, in the same direction and aligned with each other
    - Export polygons to shapefile

- Assess accuracy
    - Load automated and manual shapefiles in ArcGIS
    - Create difference features (TP, TN, FP, FN) using the intersect & difference tools
    - Check the areas using the statistics of the *Shape_Area* field
    - Create confusion matrices
    - Calculate corresponding accuracy metrics

8.6 Detailed Overview of Software

Table 6: An overview of all the software used, and what it was utilized for.

Software	Official Description	Our Application
LASTools	A collection of highly efficient, batch-scriptable, multicore command line tools. Tools to classify, tile, convert, filter, raster, triangulate, contour, clip, and polygonize LiDAR data.	· Decompressing LAZ to LAS · Cropping data to study area · Converting LAS to CSV
CloudCompare	CloudCompare is a 3D point cloud (and triangular mesh) processing software.	· Visualizing point clouds · Downsampling point clouds

Table continues on the next page..

Table 6: An overview of all the software used, and what it was utilized for.

Software	Official Description	Our Application
ArcGIS	Use ArcMap, ArcCatalog, ArcGlobe, or ArcScene, to create maps, perform spatial analysis, manage geographic data, and share your results.	· Create maps · Manually delineate linear elements · Analyse accuracy of delineation of linear elements (create difference map, assess area TP/TN/FP/FN)
Python		· Programming language
Spyder	Spyder is a Python development environment with a lot of features.	· Integrated Development Environment (IDE)
NumPy	NumPy is the fundamental package for scientific computing with Python.	· Array objects · Linear algebra (matrix operations, eigenvalues & eigenvectors) · Covariance matrix
SciPy	SciPy library provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.	· Constructing and querying k-d trees
pandas	pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.	· DataFrame objects · Read & write CSV files · Creating confusion matrices
sklearn	scikit-learn, Machine Learning in Python	· Decision tree classifiers · Creating cross validation folds · Creating parameter grids for optimization · Accuracy metrics (ROC-AUC, MCC)
imblearn	imbalanced-learn is a python package offering a number of re-sampling techniques commonly used in datasets showing strong between-class imbalance.	· Accuracy metrics for imbalanced datasets (Geometric mean)

Table continues on the next page..

Table 6: An overview of all the software used, and what it was utilized for.

Software	Official Description	Our Application
matplotlib	Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hard-copy formats and interactive environments across platforms.	· Creating plots and charts
shapely	Shapely is a BSD-licensed Python package for manipulation and analysis of planar geometric objects.	· Polygon geometry object · Converting triangles to polygon
fiona	Fiona is OGR's neat and nimble API for Python programmers.	· Export polygons to shapefile

8.7 Exercise Point Cloud Classification

An assignment to introduce classifying a point cloud in python was made to facilitate a continuation of this research by other researchers. This assignment follows on the next pages.

Classifying a point cloud with a random forest using neighbourhood parameters in python

Chris Lucas

April 21, 2017

Objective 1. Set up python, including libraries and the Spyder IDE.

To set up python we use a Python distribution which comes with a lot of libraries pre-installed called Anaconda. Anaconda is made with data science in mind and already contains many of the libraries we need.

Assignment 1. *Download and install Anaconda.*

Note. You can choose to install Python 2 or Python 3. I use Python 2 (because software packages like ArcGIS use python 2), but as for as I know the exercise should also work in python 3.

While Anaconda comes with many libraries already it does not always have every one needed. For this Anaconda comes with a package manager called conda. To use conda open a Anaconda command prompt (found in *Start Menu* > *All Programs* > *Anaconda* > *Anaconda Prompt*) and type '*conda*' followed by a command. The command to install a new package is: '*conda install <package name>*'. Generally it is a good idea to Google '*conda <package name>*' to find the most up to date version of the package available through conda. Sometimes multiple versions of a package are available through conda from different sources. Search through the Google results to find the most appropriate one for your needs and copy the text found under '*To install this package with conda run:*'.

Assignment 2. *Use Conda to install the 'Shapely' library.*

Anaconda also comes with an IDE (integrated development environment) called Spyder. Spyder (**S**cientific **PY**thon **D**evelopment **E**nvi**R**onment) is specifically made with scientific computing in mind and as such its interface is very similar to MATLAB. It features among other things an editor with syntax highlighting and code completion, a variable explorer and a debugger.

Assignment 3. *Start up the Spyder IDE and familiarize yourself with it.*

Objective 2. Download airborne LiDAR data and prepare it for processing in Python.

We will use the AHN3 (Actueel Hoogtebestand Nederland) dataset for our LiDAR data. This is a very recent dataset (it is actually still being scanned), which has a high point density and information on intensity and multiple returns. It is available at https://www.pdok.nl/nl/ahn3-downloads.

Assignment 4. *Download an AHN3 point cloud data tile.*

LAS is the conventional format airborne LiDAR data is stored in. This data is in zipped LAS format (LAZ) though and to use it we first need to decompress it. The tool available for this task is called LASTools and can be downloaded from https://rapidlasso.com/lastools/. These tools are partly free and open source software (FOSS) and partly licensed. Careful when using the paid tools without a license, because LASTools will slightly distort the output after a certain point limit. You will be notified through the console when this happens. For now we don't have to worry about this, since the tool we are going to use (laszip) is FOSS.

Assignment 5. *Download LASTools and open laszip.*

You can decompress the entire tile if you wish, but it will result in a very large LAS file. Often a research area is smaller than the tile. The output extent can be specified in laszip under *clip input*. There are two output formats available: LAS and ASCII. For processing LiDAR data in programs like ArcGIS the LAS format is needed. When using the data in Python it is easier to work with ASCII.

Assignment 6. *Use laszip to decompress the point cloud data.*

Objective 3. Visualize the point cloud using CloudCompare.

Before we are going to use Python to process the point cloud data we want to inspect the point cloud to see if downloading and decompressing the data went well and to get a good overview of the data we are dealing with. A good FOSS program to visualize a point cloud (among many other very useful tools) is called CloudCompare. The software is available for download at http://www.danielgm.net/cc/.

Assignment 7. *Download CloudCompare and load in the point cloud.*

Because CloudCompare uses 32-bit floats to store the coordinates it can lose precision when using large coordinates (which is often the case when dealing with georeferenced point clouds, like AHN3). To avoid a loss in precision CloudCompare proposes to shift the coordinates temporarily so that the coordinates are closer to zero. When saving the point cloud the coordinates will be shifted back. It is advisable to use this shifting.

Assignment 8. *Explore the point cloud in CloudCompare.*

In CloudCompare you can also visualize the different properties of the points, like the *intensity* and the *number of returns*. CloudCompare calls these scalar fields.

Assignment 9. *Visualize the intensity and the number of returns. Play around with different color scales and different display ranges (drag around the sliders in the display ranges tab).*

Assignment 10. *Have a look around at what other functionalities CloudCompare has.*

Objective 4. Import the point cloud into Python and do some basic computation.

Note. In Python we will use a range of libraries. A short description will be provided, but for detailed instructions use the library documentation and Google.

To read in the very large ASCII file we will use the pandas library (comes already installed with Anaconda). Pandas is a library that provides high performance and easy to use data structures.

Assignment 11. *Import pandas and use it to read in the point cloud csv file.*

Numpy (comes already installed with Anaconda) provides python with important functions and objects for scientific computing. One of the foremost functions it provides is the array object.

Assignment 12. *Import numpy and convert the X, Y and Z coordinates in the pandas DataFrame of the point cloud to a $N\times 3$ numpy array.*

Using these two formats we can do basically any calculation with the points necessary.

Whether a point is a last return or not is a useful property of a point. To check this we can compare the return number with the number of returns.

Assignment 13. *Create a new column in the DataFrame called last_return. Loop through the points and set last_return to true for the point where the return number is equal to the number of returns.*

Note. Looping through the points is by no means the fastest way to compute this, but it's a good exercise for when looping through points is going to be necessary. If you want also try to compute this without looping.

Note. If your point cloud is very large and the computation time gets annoyingly long just compute it for the first n points.

Objective 5. Use a k-d tree data structure to compute nearest neighbours.

When processing point clouds it's often needed to compute the nearest neighbours of a point. For example when computing neighbourhood parameters or when using a region growing algorithm. To efficiently compute these nearest points a data structure can be used. Different data structures exist, including the k-d tree, octree and R-tree. The most efficient data structure depends on the data and the application. We will us a k-d tree, but this is certainly not the only option.

The python library SciPy (comes already installed with Anaconda) contains many functions for scientific computing. It includes a spatial module, which contains an algorithm for the construction of k-d trees.

Assignment 14. *Import cKDTree from SciPy and use it to construct a k-d tree for the point cloud.*

Note. SciPy also has a KDTree function. The difference between cKDTree and KDTree is that KDTree is coded in pure python while cKDTree is coded in Cython (a version of python which gives python-like code C-like performance). Consequently cKDTree is significantly faster than KDTree.

Now that the k-d tree is constructed it can be queried for neighbours. This can be done in two ways: (i) loop over the points and query for each point separately, and (ii) query every point at once. Generally the former is more memory efficient, while the latter is more CPU efficient.

Before we query for neighbours we need to define our neighbourhood. This can be done in three ways: (i) k nearest neighbours, (ii) spherical, and (iii) cylindrical.

Assignment 15. *Use the k-d tree to compute the neighbourhood of a point using the three different neighbourhood definitions.*

Objective 6. Use a structure tensor to compute neighbourhood parameters.

Use a k nearest neighbours neighbourhood. We are going to use the point neighbourhoods to describe the points in relation to surrounding. These parameters can then be used for a classification.

We will start with a few basic geometric properties of the neighbourhood:

- Height difference:

$$\Delta_{Z_i} = \max_{j:\mathcal{N}_i}(q_{Z_j}) - \min_{j:\mathcal{N}_i}(q_{Z_j}) \tag{1}$$

- Height standard deviation:

$$\sigma_{Z_i} = \sqrt{\frac{1}{k}\sum_{j=1}^{k}(q_{Z_j} - \overline{q_Z})^2} \tag{2}$$

- Local radius:

$$r_{l_i} = \max_{j:\mathcal{N}_i}(|p_i - q_j|) \tag{3}$$

- Local point density:

$$D_i = \frac{k}{\frac{4}{3}\pi r_{l_i}^3} \tag{4}$$

where p_i is the current point which is part of a point cloud P (a set of points $\{p_1, p_2, \ldots, p_n\}$), $\mathcal{N}_i$ is the neighbourhood of p_i with points $\{q_1, q_2, \ldots, q_k\}$, where $q_1 = p_1$.

Assignment 16. *Calculate these properties for a neighbourhood.*

To further describe the surface of the neighbourhood we will use a structure tensor.

Assignment 17. *Read what the structure tensor is and how to interpret it on Wikipedia at* https://en.wikipedia.org/wiki/Structure_tensor. *More specifically what is written in the 3D structure tensor section* https://en.wikipedia.org/wiki/Structure_tensor#The_3D_structure_tensor.

So to compute this structure tensor we need to compute the eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ and eigenvectors of the covariance matrix of the points in the neighbourhood.

Assignment 18. *Use Numpy to compute the eigenvectors and eigenvalues of the covariance matrix of a neighbourhood. Tip: use the numpy.cov and numpy.linalg.eig functions.*

Note. The resulting eigenvectors and eigenvalues are not yet sorted. Sort them both based on the value of the eigenvalues, so that $\lambda_1 > \lambda_2 > \lambda_3$

Now we have everything we need to compute the following eight structure tensor features:

- Linearity:
$$L_\lambda = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{5}$$
- Planarity:
$$P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{6}$$
- Sphericity:
$$S_\lambda = \frac{\lambda_3}{\lambda_1} \tag{7}$$
- Omnivariance:
$$O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3} \tag{8}$$
- Anisotropy:
$$A_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1} \tag{9}$$
- Eigenentropy:
$$E_\lambda = -\lambda_1 \ln(\lambda_1) - \lambda_2 \ln(\lambda_2) - \lambda_3 \ln(\lambda_3) \tag{10}$$
- Sum of eigenvalues:
$$\sum_\lambda = \lambda_1 + \lambda_2 + \lambda_3 \tag{11}$$
- Local surface variation:
$$C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \tag{12}$$

Assignment 19. *Compute the eight structure tensor features for a neighbourhood.*

Assignment 20. *Compute all neighbourhood parameters for all points in the point cloud.*

Note. For now don't use a point cloud with too many points. The calculation of parameters can take a long time.

Objective 7. Classify the point cloud using the parametrized points.

Now that we have all these extra parameters to describe each point we can use this to classify. Classification can be done by a range of machine learning algorithms. A good python library for machine learning algorithms is called scikit-learn (comes already installed with Anaconda). This library contains functions for Support Vector Machines (SVM) and Random Forests (RF) and many more.

To use a machine learning algorithm you need training and testing data to feed the machine. This can be pretty labour intensive to get. CloudCompare can be a good tool for this.

Assignment 21. *Use CloudCompare to manually segment a part of your point cloud into different classes (for example buildings, vegetation, ground, water, etc..). Tip: Use the segment tool to segment points belonging to a class from the main point cloud. Use the merge tool to merge all segments of a class together. Export the resulting point clouds separately per class.*

Assignment 22. *Import the class point clouds into python and merge them into one pandas DataFrame with a 'class' column containing the designated classes.*

We are going to use a random forest for classification. A random forest is an algorithm that creates a range of decision trees based on random subsets of the data. These decision trees will all classify and the class that gets the most votes from the trees is used as the final classification.

Assignment 23. *Read up on random forests in python, for example on* http://blog.yhat.com/posts/random-forests-in-python.html

Assignment 24. *Train a random forest classifier (from scikit-learn) with train data and use it to classify test data.*

To assess the accuracy of a classification we can make a confusion matrix, which shows the actual classes against the predicted classes. Using this matrix we can calculate different scores which describe the performance of our classifier.

Assignment 25. *Create a accuracy assessment using a confusion matrix and corresponding precision, recall, accuracy, F1 and kappa scores.*

Another method to assess the performance of a classifier is the ROC-curve. This curve can be made for each class. It shows the true positive rate plotted against the false positive rate at various decision thresholds. The area under a ROC-curve (AUC) is a measure for the performance of the classifier for that class.

Assignment 26. *Create ROC-curves for the different classes and compute the AUC.*

Now that we have an idea of the performance of the classifier we can either use it to classify the rest of the points or if we are not satisfied with the accuracy we can try to improve the classifier first. This can be done by either changing the parameters of the classifier (for example the number of trees in the random forest), by using a different set of point features (for example by checking the feature importances and leaving out unimportant features), by improving the manually classified point clouds (if they are not yet perfect), or by using a different machine learning algorithm altogether.

Assignment 27. *Make a classifier you are satisfied with and use it to classify the entire point cloud.*

Assignment 28. *Export the point cloud and visualize the classification (in CloudCompare, ArcGIS or some other software). Explore the results and visually check if the classification results seem good.*