Classification and Identification of Small Objects in Complex Urban-Forested LIDAR data using Machine Learning

[a]William F. Basener, [b]Abigail Basener

[a]Mathematician, Spectral Solutions
Faculty, University of Virginia, Charlottesville, VA
Professor Emeritus, Rochester Institute of Technology, Rochester, NY
[b] Tall Oaks Academy

ABSTRACT

Classification in LIDAR data is the process of determining points on terrain types and objects, often with the goal of determining land use and/or building footprints. In this paper we endeavor to classify terrain types and objects at a high level of detail in a complex scene that includes buildings, forested areas, and steep hillsides. Our object classes include buildings, building rooftop structures, forest trees, landscape trees, landscape bushes, cars, light posts of varying sizes, fences, paved surfaces, and grass. Our classification method of choice is a Random Forest, but we also investigate other machine learning methods including K-Nearest Neightbors and Linear Discriminant Analysis. We evaluate the effectiveness of the algorithms for accuracy, required training sample size, and runtime.

Keywords: lidar, ladar, remote sensing, classification, identification, building extraction, object classification, terrain classification

1. INTRODUCTION

The goal of this paper is to present results for using machine learning to classify and identify objects in LiDAR data over a complex scene with urban and forested areas over rolling terrain. As a second goal, we provide a detailed exposition of a random forest classifier, including the tuning (sometimes called optimizing) the parameters and using methods that provide insight into the structure of the random forest. We provide a test of the random forest in comparison to a limited selection of other classifiers. Our focus is to provide a very thorough understanding of random forest, which is proven a consistent top-performer in large tests,[1] not to provide a test to demonstrate superiority in accuracy over all algorithms.

Our classes are chosen to make the task difficult, with some classes being very similar such as sports turf and road surfaces. We also have classes that are small, consisting of a single grid cell, such as individual people and light posts. While our task could be called classification, we at times use the term identification to stress the goal of identifying between these sometimes small and very similar classes.

The data collected by a LiDAR system consists of many points with $x - y - z$ coordinates, often called a point cloud. Each point in the cloud may have additional information, for example a time at which the point was measured a or spectra. The primary goal of this paper is to use machine learning methods to classify objects in LiDAR data collected over a complex scene over rolling terrain. A secondary goal is to provide an exposition of a fully developed random forest classifier, using all the standard methods for optimizing a random forest.

Machine learning is the general class of algorithms that will learn to perform a task given a set of training examples of the task. The task is typically assigning data observations to known classes (called classification) or assigning a number value to data observations (called regression - for example assigning a value to houses based on square footage and age). We emphasize the random forest classifier because in extensive tests show it is one of the best performing algorithms and because it has particular rigorous justification for use in data like LiDAR where features can have different units and the classes do not generally have a normal distribution.

The LiDAR data was collected over the Rochester Institute of Technology campus on October 13, 2006. The dataset contains approximately 17×10^6 detected pulses. The .las file contains x, y, and z coordinates for each pulse but no spectral information. The class map for our data along with the max height for each cell is shown in

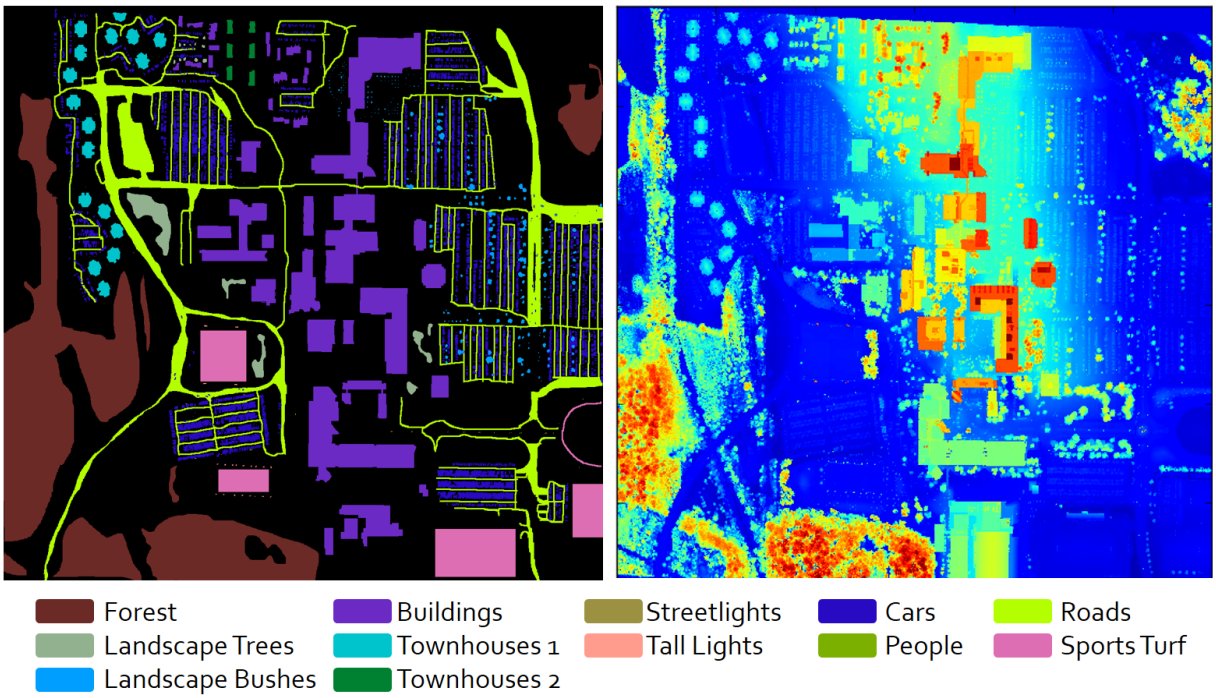

Figure 1. The class map for our data (left with class labels below) and the maximum height above ground for each cell (right) are shown.

Figure 1. Observe that there is high terrain in the upper-center portion of the image. This higher altitude region is more than 30 meters above the lower regions. Also observe that there are forested areas in the upper-right and down the left side of the image. The variety of objects and terrain height make this a complex scene.

As a pre-processing step, the points with z-coordinate outside the 0.1 to 99.9 percentile range were removed. The points were binned in the $x - y$ coordinates to a grid with 1150 rows and 1200 columns, resulting in a cell size just under one meter by one meter and an average of 12 points per cell. There are points near the edges of the image that had no data, and these were filled in with nearest neighbor averaging. Because the point cloud is collected in a series of overlapping scans, there is variation in the number of pulses in each cell. Over 86% of the cells have at least 5 pulses and the standard deviation of the number of points in each cell is 12. For nearly all cells in the areas of interest in the image, this resulted in sufficient number of pulses to obtain a meaningful min height, mean height, and max height for each cell.

To create ground truth for the gridded image, we manually created a ground truth image with the same dimensions as the gridded LiDAR image, and where the color of each point in the truth image indicates the class for the associated cell. Classes were selected so that some amount of separation between classes would be expected but correctly separating all classes would be very difficult.

We created three vegetation classes. The Forest class consists of vegetation naturally occurring around the campus. The Landscape Trees and Landscape Bushes classes are trees and bushes planted around the campus. The difference between trees and bushes was made visually from the imagery, and there are pixels that could reasonably assigned to either class. We have three classes of buildings. The Townhouses 1 and Townhouses 2 are two different types of student housing buildings. The general Building class consists of all other buildings. We have two classes of outdoor lighting. The Streetlight class consists of lights that are in parking lots and along streets. The Tall Lights class contains lights over sports areas, including a soccer field, a football field with track, and tennis courts. The Cars class consists of cars and car-sized trucks. The People class consists of individual pixels that seemed to be a person (appropriate height, width, and in pedestrian areas). We used two flat surface classes. The Road class is roads, and the Sports Turf class consists of tennis courts and athletic fields.

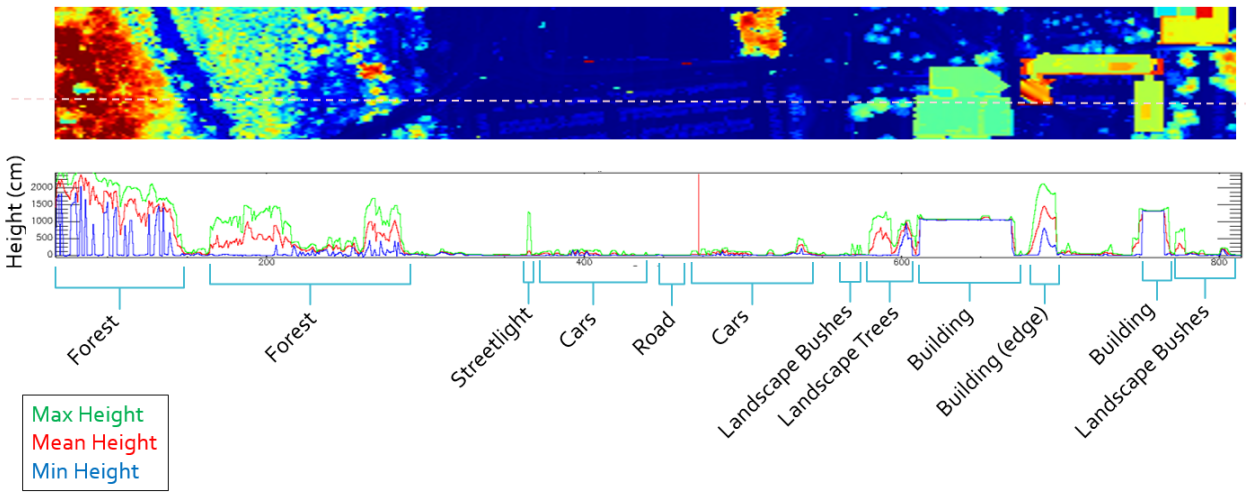

Figure 2. A cross-section of the data set showing the minimum, mean, and maximum height LiDAR return, with examples of objects from our classes of interest. The height shown is after a digital elevation model was subtracted.

2. MACHINE LEARNING

In this section we provide an overview of machine learning with an emphasis on the random forest algorithm. Machine learning is process of teaching a computer to learn to perform complex tasks, usually classification or regression. The computer learns from a training set of data and attempts to apply its acquired ability to additional data. The term machine learning emphasizes that we are enabling the computer to learn from observations instead of coding hard rules for the desired task directly.

There is a diverse variety of machine learning algorithms for classification and regression. The most important metric in evaluating a machine learning algorithm is accuracy, although there are multiple methods for accuracy, for example percent of observations correctly assigned to their class (percent accuracy) or area under a ROC curve (AUC). Other evaluation metrics that can be important are the number of training samples required and runtime. An ideal algorithm can achieve high accuracy using a moderately small training sample in a short runtime.

The standard notation for the classification task is as follows. For a training set of feature data $X = \{X_1, X_2, ..., X_N\}$ (where each X_i is the column vector of feature values for observation i), we write the class label for the i^{th} observation as Y_i. The goal in classification is for a new observation O (a new feature vector), we would like to correctly determine the class that O belongs to. Treating the data X as a set of points in n-dimensional space, the surfaces that separate the classes are called decision surfaces. The learning algorithm creates a function $G : \mathbb{R}^n \to \{C_1, ..., C_m\}$ (where m is the number of classes) that approximates the relations in the training data. Each machine learning algorithm has a set of rules for determining the function G and how to tune those rules based on the observations in X.

Training a machine learning algorithm involves creating a set of rules from the training data that will generalize robustly to additional observations. When an algorithm learns rules that represent relations between features and classes in the training data but not in general (for example, trying to develop a rule based on a single outlier data point), we say that the algorithm is overtraining. Increasing the size of the training data set will help avoid overtraining; as the algorithm has more observations to learn form, it can determine which rules are robust. An important criteria for a well-chosen algorithm is the ability to train, without overtraining, in a limited training data set. For this reason we will evaluate each learning algorithm for accuracy over a range of training data sizes.

In this section we give a brief overview of the machine learning algorithms used in this paper.

2.1 Linear Discriminant Analysis Classifier

For the linear discriminant analysis classifier (LDA), we assume the distribution of feature values for each class is modeled by a multivariate normal probability distribution. The distribution is estimated from the data assuming that every class has the same covariance but a different mean, and then the function G assigns an observation O to the class that gives the maximum likelihood. With the Gaussian assumption, the would mean we compute a Covariance matrix for each class, and then assign an observation O to the class with the minimal Mahalanobis distance.

A linear discriminant analysis classifier is Bayesian optimal when the classes are given by the assumed distributions, and Bayesian assumptions enable computation of probabilities with the class assignments. However, LDA can suffer egregiously when this distribution assumptions is not satisfied. It is worth noting that a LDA will always have linear decision surfaces.

2.2 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a conceptually simple algorithm that has potential high accuracy for complex nonlinear decision boundaries. For a training set of feature data $X = \{X_1, X_2, ..., X_N\}$, with the class label of X_i being Y_i, if we are given a feature vector for a new observation O we:

1. Determine the K elements from X that are closest to O in some metric, called the K nearest neighbors for O.
2. Assign the predicted class $G(O)$ to O to be the class that is most common among the classes for the K nearest neighbors.

The value of the parameter K can vary between implementations of KNN, as well as rules for breaking ties among the nearest neighbors and rules for "most common" often weighting the nearest neighbors by distance to O and taking a weighted vote. Small values of K permit the function G to learn complex contours around the training data but have the drawback that they are susceptible to overtraining. Larger values of K result in a function G that is less susceptible to overtraining but averages out complex relationships in the training data and thus require more training points to learn complex contours. A common rule of thumb is to choose K to be the square root of the number of training dat points. In general KNN has the ability to learn complicated decision surfaces, but generally requires a large number of data for training especially in high dimensions.

2.3 Classification and Regression Trees (CART)

A classification tree is a decision tree - a set of threshold-based decisions that recursively split the data in to subsets. A simple example showing the classification of buildings, light posts, and cars using the height and width of each object as features is shown in Figure 3. To apply this decision tree to an object O, we check if the height of O is greater than 3 meters. If it is not, we label O as a car. If the height is greater than 3m, then we check if the width. If the width is less than 1m, then we label O a light post. If is is greater than 1m, then we label O as a building. A significant advantage to decision trees is that they to be human interpretable. However, the simple split criteria also restricts the type of separation between classes, limiting the accuracy of decision trees.

A classification tree is trained from the training data as follows:

1. Determine a feature f and a threshold t such that subdividing the training data into two subsets (one subset of all observations with $f > t$ and one with all observations with $f < t$) based on the criteria that the subsets should be as "pure" (in the sense of not splitting class members across the subsets) as possible. (The most common metrics for purity of the subsets are Gini index, which we use, and entropy.)
2. Repeat Step 1 applied to the subsets until either further subsetting provides insufficient improvement in subset purity or until all remaining subsets are of some minimal size.

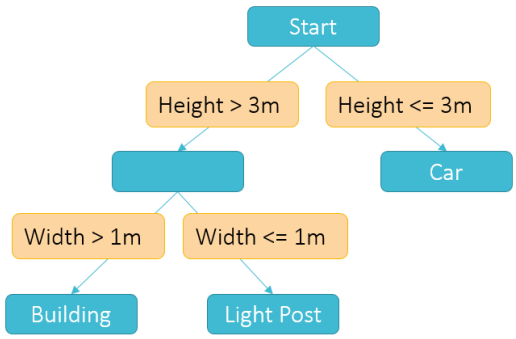

Figure 3. An example decision tree for distinguishing buildings, light posts, and cars.

The parameters that need to be chosen are the measure for subset purity, the minimal leaf node size, and the minimal improvement in purity required for further subsetting. The Gini index is measure of impurity, which is between 0 (when both subsets have equal amounts of each classes) and 1 (when each subset after the split contains just one class). The nodes in the tree at which a split is made are called branch nodes, or just nodes. The nodes at the end of the tree where the classification assignment occurs are called leaf nodes, or just leaves.

For simple classification tasks, where the range of feature values that separate classes are well-defined, decision trees can be very accurate. For our simple buildings - light posts - cars examples, the provided decision tree would work perfectly if the light posts are standard tall parking lot posts. But if we try to us the decision tree shown in Figure 3 to classify lights for walking paths that may be less than 3m tall or very wide wide stadium lights, it would clearly fail. Or, if our cars class also includes trucks of various sizes, again we would likely get errors. It would be possible to create a decision tree that will perfectly classify every point in a training set by simply growing the tree (that is, creating additional node) until every leaf node contains members of just one class. (This is assuming no two observations in different classes have the exact same feature values.) But adding nodes (called growing the tree) until every leaf is a single class leads to rules based on a small number of examples; overtraining. For this reason, a tree is usually only grown (that is, nodes are only recursively added) until some some limiting criteria is met.

2.4 Random Forests

A Random Forest classifier is a collection of simple classification trees, where each classification tree is trained using a random subset of the training data and the split at each node is created using a random subset of features. Each tree in the forest gets a "vote" on the class to assign to a new observation, and the class assignment made by the Random Forest is the tally of the votes. Usually this means the class that the Random Forest assigns to the new observation is the class is is assignment most often among the trees, but in some variations the votes from different trees can be weighted differently. The votes from the single tree classifiers can be tallied to determine probabilities; the probability that an observation O belongs to a given class is the fraction of single trees that assigned the observation to that class. An exceptional practical introduction to random forests is given in a Microsoft technical paper,[2] in which the authors use random forest to recognize human body positions in their XBox Kinect sensor system.

There is theoretical justification that a random forest will retain much of the accuracy of decision trees while the random variation among the trees learns net rules that generalize.[3,4] This is supported in practice.[3] The repeated random sampling of data and features results in rules that retain positive aspects of decision trees but reflect general rules rather then idiosyncracies of the particular observations in the training data. In tests on a variety of data, random forests consistently among the most accurate algorithms.[1]

Random forest had a build in form of validation that is useful for optimizing parameters. The process of selecting a random subset of data to build an individual tree is called *bagging*, and the portion of the original data that was not selected for training is called the *out of bag sample*. For each observation, we can determine

the output of all trees for which the observation was in the out of bag sample to determine a class. The error from these classes (number of correctly classified observations/number of observations) is called the *out of bag error*. The out of bag error is a very good measure of the accuracy of the random forest since the testing is effectively done on observations that were not used in training. Typically the out of bag error will be a little higher than the error on test data (because the out of bag error is computed from a subset of the trees). To determine the value for any of the parameters, random forests are created using sequence of values and the forest with the lowest out of bag error is used to determine the parameter value.

The primary parameters that need be chosen are the number of trees (more trees improves accuracy up to a point), the fraction of data to be used to build each tree (a smaller fraction results in more variation among trees), and the number of features to select for building each node. Practice has shown that there is usually a broad range of values for any of these parameters that results in good performance, and when needed parameters can be chosen by repeated testing and optimization. However, there are some guiding principles. The number of trees can be determined by building random forests with a continuum of values for the number of trees and plotting the out of bag error as a function of the number of trees; as more trees are used the error decreases and then levels off, and the number of trees is chosen to be a little past the point of leveling off. A rule of thumb for the number of features to select is the square root of the total number of features, but as before the number of features can be optimized by building multiple random forests with different values, and choosing the value for the number of features. While random forest are generally robust with respect to these features, choosing too few trees results in less than optimal accuracy, and using too many has a (usually minor) computation cost. Using too many features results in potential overtraining, while too few can result in less than ideal accuracy. Choosing a fraction of data for building each tree that is too large can result in overtraining (because there is not sufficient variation among the trees), and choosing a fraction of data that is too small results in a high required number of trees for optimal accuracy. Theoretical justification for these is not completely understood.[5] In the initial theoretical paper,[3] the author recommends growing trees fully (relying on variation among trees to prevent overtraining), although subsequent papers recommend using stopping criteria. Selecting stopping criteria also can be optimized by testing with the out of bag error.

3. METHODS

In this section, we describe the methods for computing a set of features for each cell in the gridded LiDAR data and then show the parameter optimization for the random forest. The goal in computing features is not to necessarily determine features that are perfect indicators for specific classes, but to determine features that give some indication of classes that can be used by classification algorithms to determine class rules.

3.1 Features

As a pre-processing step, we applied a building-detection algorithm to identify building, computed a digital elevation model (DEM) using the non-building cells, and subtracted the DEM from the data to get an elevation-removed dataset. The method of building detection and DEM construction are not of particular important; what is important is that the classification will be done using imperfections that are inevitably introduced by such standard pre-processing. For the rest of the paper, we assume that the data has been DEM-subtracted unless otherwise noted. The maximum height within each cell after subtracting the DEM is shown in Figure 4.

For each grid cell, we have a collection of z-values, or heights, for points in the cell. An indicator whether the grid cell was in the building mask (value equal to 1) or not in the mask (value equal to 0), called building_mask. The minimum height of points in the grid is called min. The mean height of points in the grid is called mean. The maximum height of points in the grid is called max. The median height of points in the grid is called median.

Local texture features are computed using a 3×3 sliding window. For each window size, we compute the mean value of the mean, min, and max features to obtain local_min, local_mean, and local_max. We also compute the standard deviation of the values of the min, mean, and max features over the window, called stdev_min, stdev_mean and stdev_max.

We computed four features that measure the "roughness" in a cell, measuring the difference between the max and min heights. The first feature, simple called roughness, is the max feature minus the min feature for each cell,

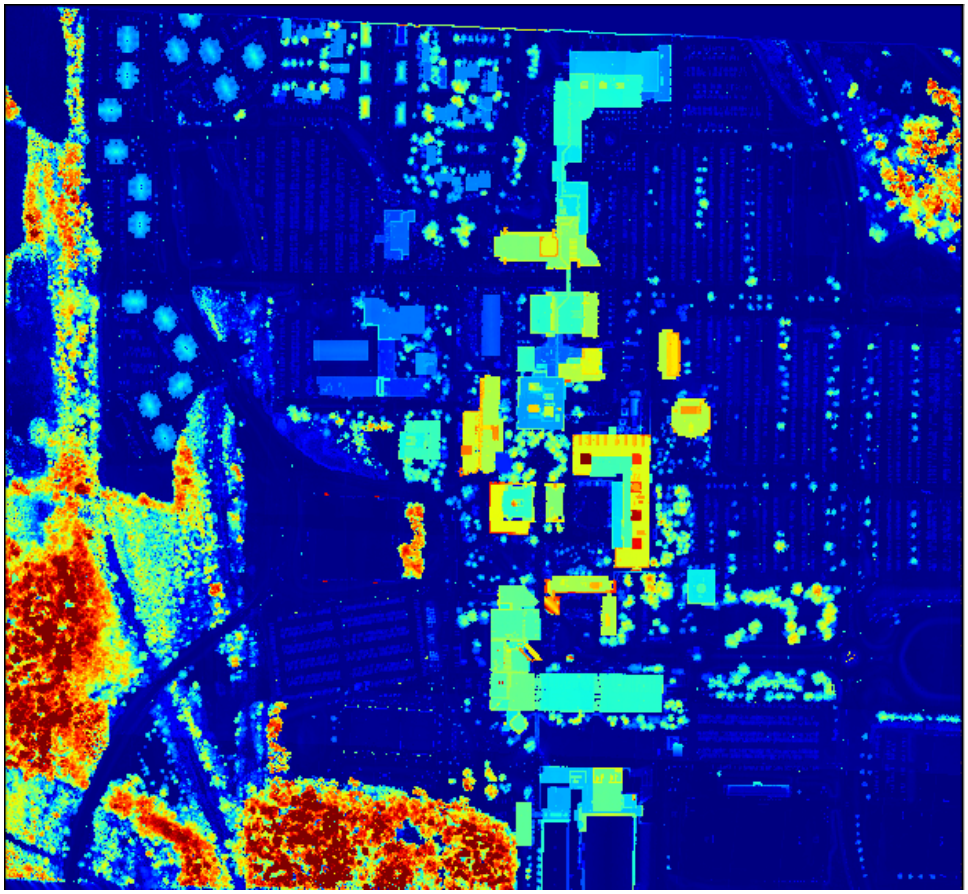

Figure 4. The maximum height above ground for each cell is shown.

and is standard in LiDAR. We computed the min feature divided by the max feature, called the min_max_ratio. (To avoid extreme values for this feature, we set the min to 0 if it was less than zero and the max to 1 if it was less than 1.) The min_max_ratio feature gives a measure of the roughness in a cell that still measures high when the max is small be still substantially larger than the min, for example with short vegetation. We also created a feature called minsqrd_max_ratio, which is the min squared divided by the max, normalized to avoid extreme values as before. We computed the minimum of the roughness over a local 3×3 window, which we call local_min_roughness. This may be useful in distinguishing building edges, which may have a high roughness value but low local_min_roughness, from trees which should have a high roughness and local_min_roughness.

The magnitude of the gradient of each cell (computed with respect to the spatial dimensions) was computer to create a feature called gradient. Similarly, the Laplacian was computed to create a Laplacian feature.

The output of the building detection algorithm was a building mask (equal to 0 if the cell is not a building, and 1 if it is a building). The mask was used as a feature, as was the local max of the mask over a 3×3 window (which we call wide3x3_building_mask) and the max over a 5×5 window (which we call wide5x5_building_mask. The expectation is that these local maximum features would help distinguish building edges from trees; both would have a high roughness score but the building edges should have nonzero scores in the local max building mask feature.

In the process of creating the building mask, we constructed a mask of edges, which we include as a feature (which we call building_edges_mask). For each cell, we also determined the component of the building max containing the cell and then used the number of cells in the component as a feature called class_size. For any cell in the building mask, the class_size is the area of the building containing the cell in square meters. Motivation

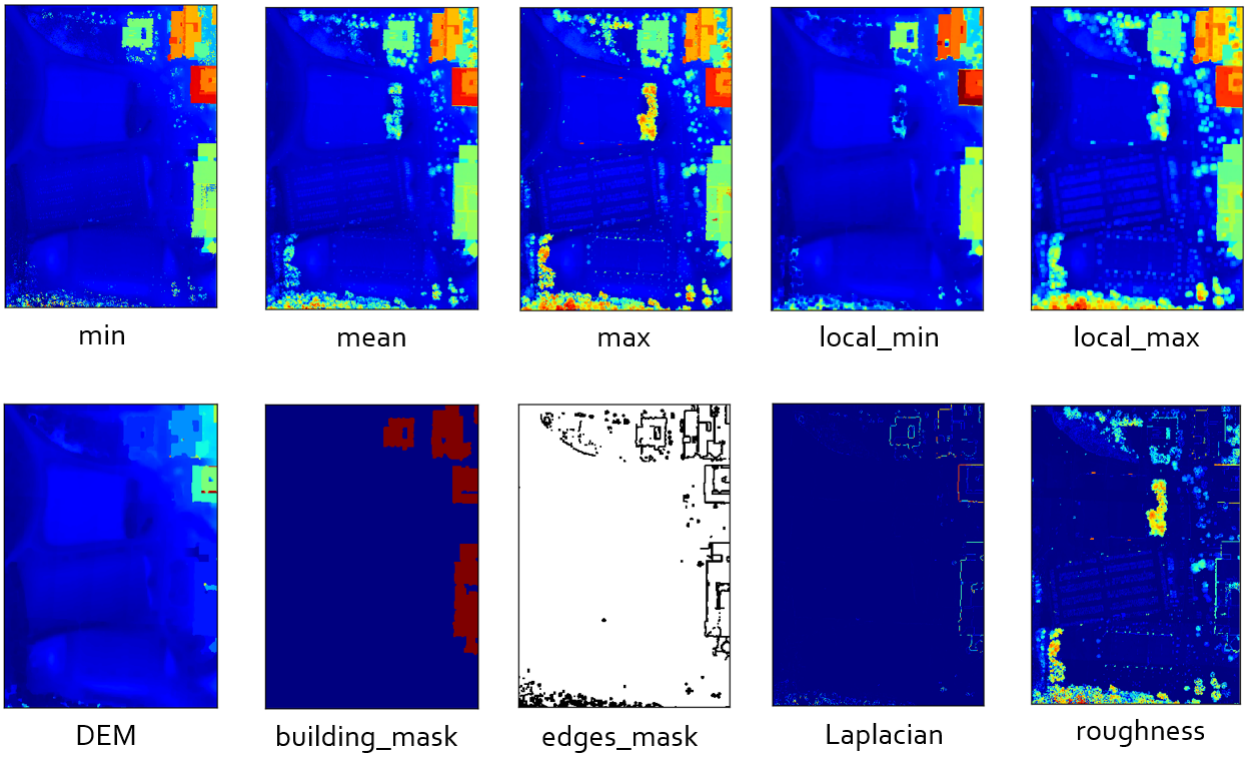

Figure 5. Feature images for a chip of the image. The region includes buildings on the right-hand side and trees in the center and lower areas. The rectangular areas are, from top to bottom, a artificial turf ball field, a parking lot with cars, and tennis courts.

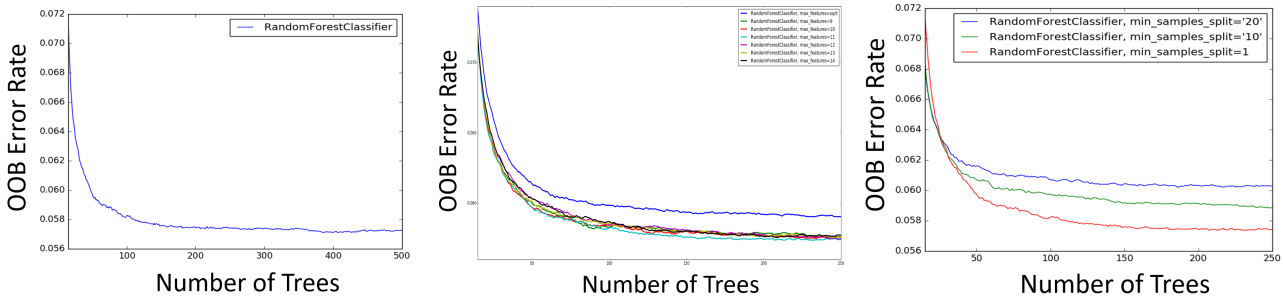

Figure 6. Plots from parameter tuning in random forest. These are the plot used for choosing the number of trees (left), the plot used to choose the number of features for each split (center), and the plot used to determine that growing trees fully (equivalently, having a minimum of 1 data point to create new split in growing) (right).

for the class_size feature is that the building mask was not perfect, and some small regions in the trees were contained in building mask, and the class_size feature may help distinguish these regions from true buildings. A selection of our features over a subset of the data are shown as image chips in Figure **??**.

3.2 Parameter Tuning

In this section we present the plots used for parameter tuning. To determine the number of trees to use in the forest, N, we build forests with N ranging from 1 to large numbers and plot the OOB accuracy as a function of the number of trees. Increasing N decreases the error at the cost requiring longer runtime so we chose N a little after the point where the error levels off. From the plot in Figure 6 we chose $N = 250$.

To choose the number of features to select in each split (called max_features), we compute random forests with varying values and choose the value that gives the lowest OOB error. For convenience we do this for random forests with various numbers of trees to create the plot shown in Figure 6. The lowest error occurred when using 11 features at each split. Similarly, we plotted the OOB error using different minimal number of data points for a split resulting in the plot shown in Figure 6. From this, we decided to use min_samples_split equal 1, which means that the trees will be grown fully, without stopping criteria.

The first two nodes for two trees from the random forest are shown in Figure **??**. The full trees have on the order of 20 layers of nodes, and so only the very initial stage is shown in each case. In the first tree (top), the features are max height and then local standard deviation. (Recall that the local standard deviation is the standard deviation of the mean height over the cells in a 3×3 window, giving a measure of the local fluctuation in mean height.) In the second tree (bottom), the features used are the local mean height and then the roughness (right-side) and building mask (left-side). These two trees use completely different starting features.

Below each lowest shown node are is a table showing the percent of data from each class that are in the subset at that node. Observe that for each class, there is a single node that contains most of the data for that class. Following the splits at the nodes, we can see the logic inferred by the random forest. For example, the leftmost node in the first tree contains data cells that have a low local height and low roughness; mainly Sports Turf and Roads. The next subset contains cells in the Cars and People classes, objects that we would expect to be short (less than 1.35m in height) and have a larger roughness (roughness greater than 0.53m, where recall that roughness is the max height minimum the min height in the cell), which makes sense. The reader is invited in inspect the rest of the subsets for similar logic.

This examples shows the result of random sampling of features; we have two trees that use completely different features and yet both do a good job of separating the data into relatively pure subsets. In choosing features for a random forest classifier, it is more important to get a variety of features, often with multiple features that measure something in different ways, than to build perfect features the single-handedly separate classes.

In Figure 8, we show a scatterplot of the classes except the forest class on the maximum height and local standard deviation features. The forest class was excluded because it spans almost the entire range of the image, obscuring the other class data. The features used are the first two features used in the first tree in Figure 7, and the thresholds from the first two nodes in that tree as shown as vertical and horizontal lines on the plot. Observe that while these thresholds separate some of the classes, there is a lot of overlap between classes evident in this plot.

Accompanying the random forest architecture is a useful measure of feature importance, called feature importance, gini importance, or mean decrease impurity.[4] It is a measure of the effectiveness of the feature for creating more pure subsets after splits using the feature. It is computed as the total decrease in node Gini index impurity (weighted by the proportion of data reaching each node split by the feature), averaged over all trees in the ensemble. Specifically, for feature X_m, if we let $p(t)$ be the proportion of data reaching node t, let $\triangle_i(t)$ denote the decrease in Gini index at the node, $f(t)$ denote the feature used for the split at node t, N_{trees} be the number of trees in the forest, then the importance of feature X_m is

$$\text{Imp}(X_m) = \frac{1}{N_{\text{trees}}} \sum_{T} \sum_{t \in T : f(t) = X_m} p(t) \triangle_i(t).$$

Figure 9 shows the feature importance for the features in our random forest. For each feature, the bar shows the mean decrease in gini index and the wiskers indicate the range of decrease (or increase) in gini index for splits using that feature. Recall that the Gini index is between 0 and 1.

4. RESULTS

In this section we present the results from our tests. In addition to the random forest explained in Section 3.2, we also tested a K-nearest neighbors classifier (KNN), a single tree classifier, and a linear discriminant analysis (LDA) classifier. All classifiers were implemented in Python using the sklearn package. The goal of this paper is to provide a well-tuned random forest classifier, which is likely to be a top performing algorithm for this type

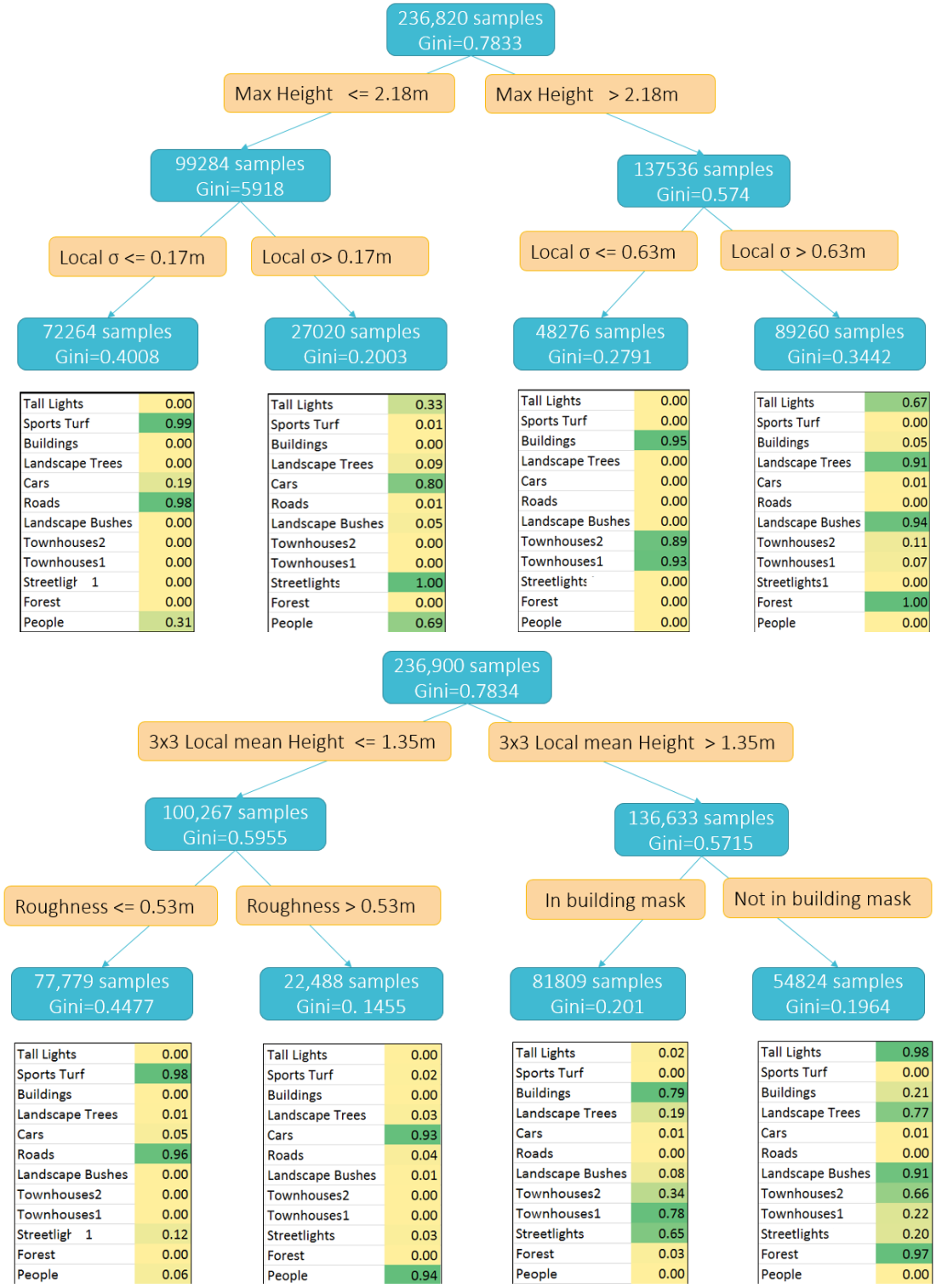

Figure 7. The top section of two decision trees from the random forest are shown, including only the top two nodes. At each split, the feature name and threshold value are shown as well as the Gini Index used to determine the split. Beneath each of the lower nodes are shown the percent of each class contained in that node.

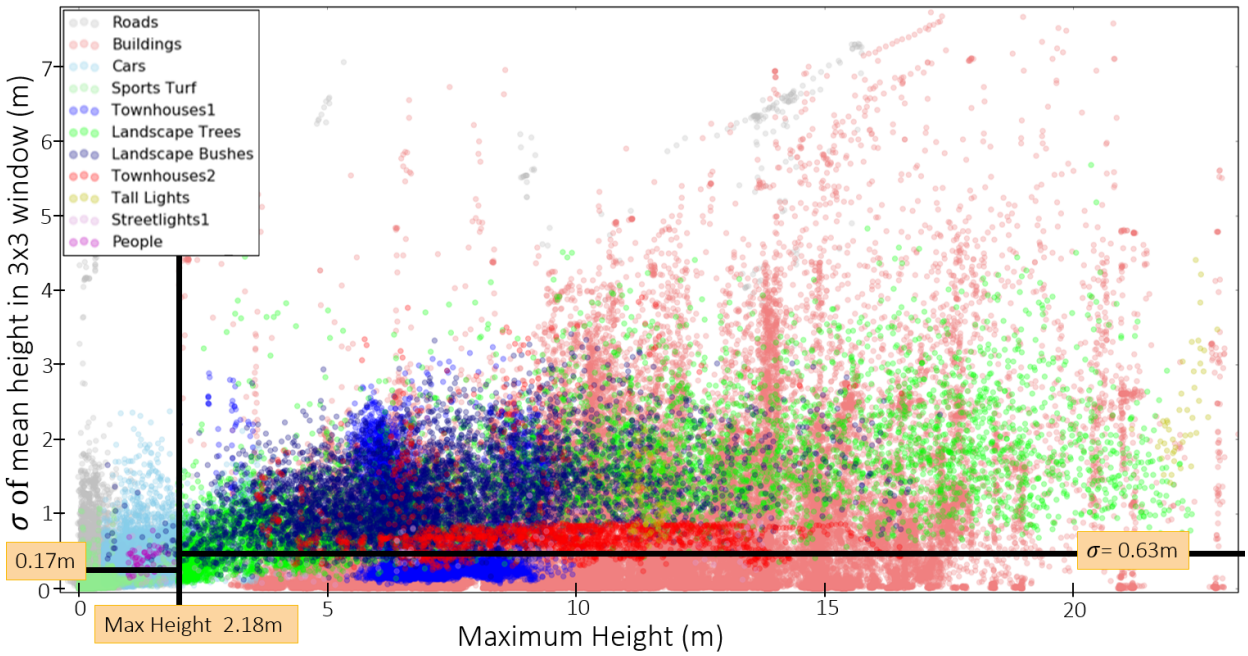

Figure 8. A scatterplot of all classes except the forest class on the maximum height and local standard deviation features. Also shown on the plot are the thresholds used for the tree in Figure 7.

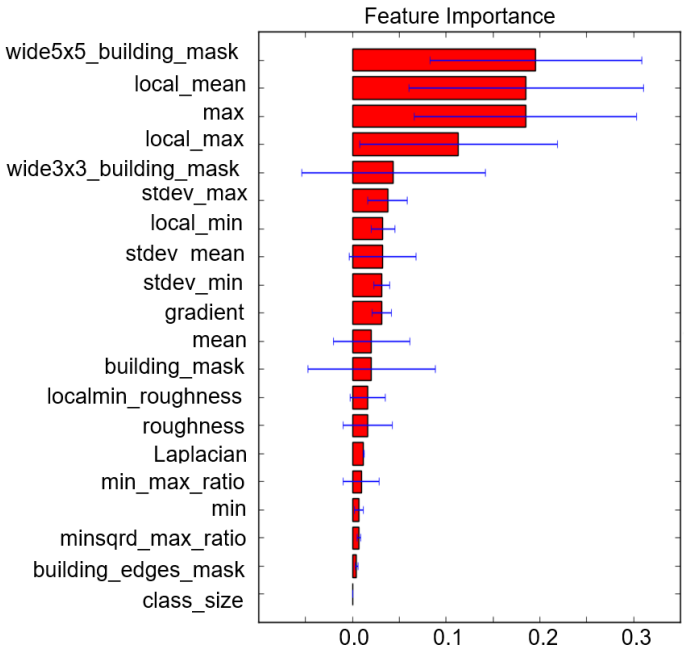

Figure 9. Importance of the features in our random forest. The feautre importance is the average decrease in Gini index over splits using the feature.

Table 1. Classification accuracy

Training Data Size	RF	KNN	Tree	LDA
10	61.1	44.6	57.6	55
50	76.2	51.7	64.4	53
100	84.5	59.0	71.3	60
1,000	84.5	70.6	79.0	58
10,000	90.5	80.8	85.0	63
100,000	95.1	91.9	91.1	74

Table 2. Runtime in seconds to train the classifier and apply the classifier to the test data.

Training Data Size	RF	KNN	Tree	LDA
10	61	2	0.1	0.3
50	76	6	0.1	0.2
100	84	9	0.1	0.3
1,000	84	26	0.3	0.3
10,000	90	58	2.1	0.6
100,000	95	37	11	1.8

of data. As such, careful tuning of parameters for the other algorithms is outside to current scope of the paper. Some form of per-feature normalization (for example, putting all features in units of z-score or normalizing the range to $[0, 1]$) could potentially help KNN and LDA. Yet one of the big advantages of random forest is that it is impervious to difference in units and categorical variables.

Our four classifiers were tested using a random sample of the data for training and the remainder of the data for testing. A stratified sampled method was used, meaning that for a training data size of N, N samples were randomly selected from each class unless N exceeded 80% of the class, in which case the sample was a random sample of 80% of the class. This was done to avoid impact of disparate class sizes. The People class had only 50 members while the Forest and Building classes have on the order of a million members. The stratified sample guaranteed that a random sampling would not cause bad classification results on a class that was under-sampled and guaranteed that each class had enough members for testing.

The only feature we had to select was the number of neighbors in KNN. We chose K to be the maximum of either 10 or the square root of the training data size (which is the number of observations for each class). The classificaiton accuracy is shown in Table 1. The computational time needed to train each classifier and run on the test data is shown in Table 1.

A goal of this paper is to determine if a random forest classifier can distinguish between very similar classes, as a form of identification. The confusion matrix for our random forest with a training data size of 100,0000 data points is shown in Table 3. The first column gives name of the actual class and the number in each column gives the percentage of that class that was identified in the class given by the label in the top row. For example, the 92% of the Tall Lights we properly classified as Tall Lights and 8% of the Tall Lights were mis-labeled as Forest. The people class was most difficult to classify, with all people cells being labeled as cars.

5. CONCLUSION

From Table 1, we see that the random forest was significantly more accurate than any of the other algorithms. Even with a modest 100 training samples from each class, it had a reasonable accuracy of 85%. The KNN classifier requires between 10,000 and 100,000 training samples to meet this accuracy. This is a general trend, that KNN can be very accurate but generally requires significantly more training samples than random forest. We can also see this trend comparing KNN to a single tree; for small training size the tress does better but for larger training sample sizes we see KNN meet the performance of the tree. We also see what should be expect poor performance in LDA, as the features have different units and the feature values for classes are not expected to follow a normal distribution. The main missing point of comparison would be a support vector machine (SVM), which is often a top performer along with random forest for data like ours. Building an SVM with a properly chosen kernel and other parameters is beyond the scope of this paper, and we decided to not include SVM for comparison instead of using a poorly built SVM that could lead to improper conclusions.

Table 3. The confusion matrix for our random forest classifier with a training sample size of 100,000.

	TL	ST	Bs	LT	Cs	Rs	LB	T2	T1	SL	F	P
Tall Lights	92	0	0	0	0	0	0	0	0	0	*	0
Sports Turf	0	65	0	0	0	23	0	0	0	0	0	0
Buildings	0	0	99	0	0	0	0	0	0	0	1	0
Landscape Trees	0	0	0	25	2	0	5	0	0	0	66	0
Cars	0	0	0	0	99	0	0	0	0	0	0	0
Roads	0	6	0	0	1	93	0	0	0	0	0	0
Landscape Bushes	0	0	0	2	2	0	62	0	0	0	33	0
Townhouses2	0	0	5	0	0	0	0	93	2	0	0	0
Townhouses1	0	0	2	0	0	0	0	0	97	0	0	0
Streetlights	0	0	0	0	0	0	0	0	0	88	0	0
Forest	0	0	0	0	0	0	0	0	0	0	99	0
People	0	0	0	0	100	0	0	0	0	0	0	0

The accuracy of 95% from our random forest trained on 100,000 data points is a promising accuracy given the similarity between some of our classes. Accuracy was by far the poorest on the People class. This may be because the People are genuinely hard to distinguish from cars using our features, in which case we could look for new features that may help distinguish People from Cars. Or, the small number of training points for People may be a source of error; classification algorithms generally optimize to minimize error and this criteria sometimes causes small, rare, classes to get missed. Since it is difficult to find many more People cells in the data, weighting the People class or replicating members of that class to boost their importance could help force the random forest to identify them; in particular there would likely be more splits that seperate People becuase the replicated People samples would have a larger impact on the Gini index than the current small number of samples.

As expected, there was some confusion between similar classes. The classified landscape trees more often as forest than landscape trees; this is obviously reasonable and there is probably significant genuine overlap between these classes. Similarly, the algorithm was only moderately successful at distinguishing roads and sports turf. Although the 93% and 65% accuracy for these classes could be considered good, given the high degree of similarity between these classes. The algorithm had little difficulty distinguishing between the different building types and light categories. Certainly, man-made objects like these with limited within-class variability can be easier to separate.

An approach that could be helpful distinguishing between similar classes would be to have an additional algorithm. For example, it could be useful to train an algorithm that is good at binary decisions such as the adaboost on all Cars and People only and be applied to make a final classification on every sample that gets called a car or person. Chaining multiple classification algorithms like this is often useful in practical machine learning applications.

It is worth noting that we used very little spatial information. It is likely that using convolutions with templates would provide valuable features and improve accuracy significantly. This could be done with a random forest or a convolutional neural network (CNN). Given recent strong performance by CNNs in imagery, it is likely that a CNN would perform better than our random forest at the cost of requiring more work to optimize and train. With a CNN, it is likely that landscape trees planted in isolated groups could be distinguishable from trees within a forest, for example. Implementation of these ideas is beyond the scope of this paper.

In this paper, we implemented a selection of machine learning algorithms for classification and identification of objects in LiDAR data. We provided an exposition of the random forest, showing how to tune the paremeters and interpret the underlying mechanism. As expected, the random forest was the top performing algorithm in our test. We observed that the algorithm had the ability to distinguish disparate classes effectively (such as trees from buildings) and had moderate success distinguishing between similar classes (such as sports turf and roads). We discussed some modifications and additional algorithms that could provide better accuracy, but the random forest algorithm had a relatively high accuracy given our difficult data and classes.

REFERENCES

1. Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, Jan. 2014.
2. Konukoglu Criminisi and Shotton, "Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning," Microsoft TechReport: https://www.microsoft.com/en-us/research/publication/decision-forests-for-classification-regression-density-estimation-manifold-learning-and-semi-supervised-learning/, 2011, [Online; accessed 3-March-2017].
3. Leo Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
4. Olshen Stone Breiman, Friedman, *Classification and regression trees*, Brooks/Cole Publishing, Monterey, 1984.
5. Biau and Scornet, "A Random Forest Guided Tour," https://www.hse.ru/data/2016/05/13/1128994669/BiauS14.pdf, [Online; accessed 3-March-2017].