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×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.*