The Journal of Open Source Software

DetecTree: Tree detection from aerial imagery in Python

Martí Bosch[1]

1 Urban and Regional Planning Community, École Polytechnique Fédérale de Lausanne, Switzerland

DOI: 10.21105/joss.02172

Software

- Review
- Repository
- Archive

Editor: Kristen Thyng

Reviewers:

- @JeffWalton-PSC
- @rmsare

Submitted: 02 March 2020
Published: 25 June 2020

Summary

Urban tree canopy datasets are important to a wide variety of social and environmental studies in cities. Surveys to collect such information manually are costly and hard to maintain, which has motivated a growing interest in automated approaches to tree detection in recent years. To that end, LIDAR point clouds are a very appropriate data source, especially given its capability to represent spatial features in three dimensions. However, collecting LIDAR data requires expensive equipment and raw datasets are rarely made openly available. On the other hand, aerial imagery is another major source of data that is able to capture a wide range of objects on Earth, including trees. Although aerial imagery depicts the spatial features in only two dimensions, its main advantage with respect to LIDAR is its greater availability.

The aim of DetecTree is therefore to provide an open source library that performs a binary classification of tree/non-tree pixels from aerial imagery. The target audience is researchers and practitioners in GIS that are interested in two-dimensional aspects of trees, such as their proportional abundance and spatial distribution throughout a region of study. These measurements can be used to assess important aspects of urban planning such as the provision of urban ecosystem services. The approach is of special relevance when LIDAR data is not available or it is too costly in monetary or computational terms.

Methodology

DetecTree is based on the supervised learning approach of Yang, Wu, Praun, & Ma (2009), which requires an RGB aerial imagery dataset as the only input, and consists of the following steps:

- **Step 0**: split of the dataset into image tiles. Since aerial imagery datasets often already come as a mosaic of image tiles, this step might not be necessary. In any case, DetecTree provides a `split_into_tiles` function that can be used to divide a large image into a mosaic of tiles of a specified dimension.
- **Step 1**: selection of the tiles to be used for training a classifier. As a supervised learning task, the ground-truth maps must be provided for some subset of the dataset. Since this part is likely to involve manual work, it is crucial that the training set has as few tiles as possible. At the same time, to enhance the classifier's ability to detect trees in the diverse scenes of the dataset, the training set should contain as many of the diverse geographic features as possible. Thus, in order to optimize the representativity of the training set, the training tiles are selected according to their GIST descriptor (Oliva & Torralba, 2001), *i.e.*, a vector describing the key semantics of the tile's scene. More precisely, k-means clustering is applied to the GIST descriptors of all the tiles, with the number of clusters k set to the number of tiles of the training set (by default, one percent of the tiles is used). Then, for each cluster, the tile whose GIST descriptor is closest to the cluster's centroid is added to the training set. In DetecTree, this is done by the `train_test_split` method of the `TrainingSelector` class.

- **Step 2**: provision of the ground truth tree/non-tree masks for the training tiles. For each tile of the training set, the ground-truth tree/non-tree masks must be provided to get the pixel-level responses that will be used to train the classifier. To that end, an image editing software such as GIMP or Adobe Photoshop might be used. Alternatively, if LIDAR data for the training tiles is available, it might also be exploited to create the ground truth masks.
- **Step 3**: train a binary pixel-level classifier. For each pixel of the training tiles, a vector of 27 features is computed, where 6, 18 and 3 features capture characteristics of color, texture and entropy respectively. A binary AdaBoost classifier (Freund & Schapire, 1995) is then trained by mapping the feature vector of each pixel to its class in the ground truth masks (*i.e.*, tree or non-tree).
- **Step 4**: tree detection in the testing tiles. Given a trained classifier, the `classify_img` and `classify_imgs` methods of the `Classifier` class can respectively be used to classify the tree pixels of a single image tile or of multiple image tiles at scale. For each image tile, the pixel-level classification is refined by means of a graph cuts algorithm (Boykov & Kolmogorov, 2004) to avoid sparse pixels classified as trees by enforcing consistency between adjacent tree pixels. An example of an image tile, its pre-refinement pixel-level classification and the final refined result is displayed below:

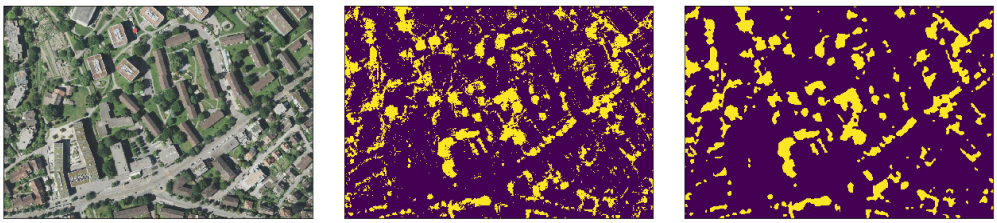

Figure 1: Example of an image tile (left), its pre-refinement pixel-level classification (center) and the final refined result (right).

Similar methods of tree classification from aerial imagery include the work of Jain, Mittal, Jain, & Sinha (2019), who follow the train/test split method based on GIST descriptors as proposed by Yang et al. (2009) but rely on the Mask R-CNN framework (He, Gkioxari, Dollár, & Girshick, 2017) instead of the AdaBoost classifier. Another approach by Tianyang, Jian, Sibin, Ying, & Jing (2018) employs a cascade neural network over texture and color features which detects single trees in a variety of forest images. Nonetheless, since the former approaches ultimately aim at single tree detection, the accuracy evaluation metrics that they provide are hard to compare with the pixel-level classification accuracy of DetecTree. The experiments performed by Yang et al. (2009) in New York achieve a pixel classification accuracy of 91.7%, whereas the example applications of DetecTree in Zurich and Lausanne achieve accuracies of 85.98% and 91.75% respectively.

The code of DetecTree is organized following an object-oriented approach, and relies on NumPy (Van Der Walt, Colbert, & Varoquaux, 2011) to represent most data structures and perform operations upon them in a vectorized manner. The Scikit-learn library (Pedregosa et al., 2011) is used to implement the AdaBoost pixel-level classifier as well as to perform the *k*-means clustering to select the training tiles. The computation of pixel-level features and GIST descriptors makes use of various features provided by the Scikit-image (Van der Walt et al., 2014) and SciPy (Virtanen et al., 2020) libraries. On the other hand, the classification refinement employs the graph cuts algorithm implementation provided by the library PyMaxFlow. Finally, when possible, DetecTree uses the Dask library (Rocklin, 2015) to perform various computations in parallel.

The features of DetecTree are implemented in a manner that enhances the flexibility of the library so that the user can integrate it into complex computational workflows, and also provide

custom arguments for the technical aspects. Furthermore, the functionalities of DetecTree can be used through its Python API as well as through its command-line interface (CLI), which is implemented by means of the Click Python package.

Availability

The source code of DetecTree is fully available at a GitHub repository. A dedicated Python package has been created and is hosted at the Python Package Index (PyPI). The documentation site is hosted at Read the Docs, and an example repository with Jupyter notebooks of an example application to an openly-available orthophoto of Zurich is provided at a dedicated GitHub repository, which can be executed interactively online by means of the Binder web service (Bussonnier et al., 2018). An additional example use of DetecTree can be found at a dedicated GitHub repository with the materials to obtain a tree canopy map for the urban agglomeration of Lausanne from the SWISSIMAGE 2016 orthophoto (Federal Office of Topography, 2019).

Unit tests are run within the Travis CI platform every time that new commits are pushed to the GitHub repository. Additionally, test coverage is reported on Coveralls.

Acknowledgments

This research has been supported by the École Polytechnique Fédérale de Lausanne.

References

Boykov, Y., & Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, *26*(9), 1124–1137. doi:10.1109/TPAMI.2004.60

Bussonnier, Forde, Freeman, Granger, Head, Holdgraf, Kelley, et al. (2018). Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th python in science conference* (pp. 113–120). doi:10.25080/Majora-4af1f417-011

Federal Office of Topography. (2019). SWISSIMAGE la mosaïque d'ortophotos de la Suisse. Available from (in French) https://shop.swisstopo.admin.ch/fr/products/images/ortho_images/SWISSIMAGE. Accessed: 3 March 2020.

Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37). Springer. doi:10.1007/3-540-59119-2_166

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *Proceedings of the ieee international conference on computer vision* (pp. 2961–2969).

Jain, S., Mittal, R., Jain, A., & Sinha, A. (2019). An efficient framework for monitoring tree cover in an area through aerial images. In *Applications of machine learning* (Vol. 11139, p. 1113918). International Society for Optics; Photonics. doi:10.1117/12.2529442

Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, *42*(3), 145–175. doi:10.1023/A:1011139631724

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), 2825–2830. doi:10.5281/zenodo.3696718

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*. doi:10.25080/Majora-7b98e3ed-013

Tianyang, D., Jian, Z., Sibin, G., Ying, S., & Jing, F. (2018). Single-tree detection in high-resolution remote-sensing images based on a cascade neural network. *ISPRS International Journal of Geo-Information*, *7*(9), 367.

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22. doi:10.1109/MCSE.2011.37

Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., et al. (2014). Scikit-image: Image processing in Python. *PeerJ*, *2*, e453. doi:10.7717/peerj.453

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 1–12. doi:10.1038/s41592-019-0686-2

Yang, L., Wu, X., Praun, E., & Ma, X. (2009). Tree detection from aerial imagery. In *Proceedings of the 17th acm sigspatial international conference on advances in geographic information systems* (pp. 131–137). doi:10.1145/1653771.1653792