AUTOMATIC CHANGE DETECTION OF DIGITAL MAPS USING AERIAL IMAGES AND POINT CLOUDS

Felix Dahle[1], Ken Arroyo Ohori[2], Giorgio Agugiaro[2], and Sven Briels[3]

[1]Geoscience & Remote Sensing, Delft University of Technology, the Netherlands - F.Dahle@tudelft.nl
[2]3D geoinformation, Delft University of Technology, the Netherlands - K.Ohori@tudelft.nl, G.Agugiaro@tudelft.nl
[3]Readar, Utrecht, the Netherlands - svenbriels@readar.com

Commission II, WG II/6

KEY WORDS: Change detection, Gradient boosting, XGBoost, Classification, Machine Learning

ABSTRACT:

In many countries digital maps are generally created and provided by Cadastre, Land Registry or National Mapping Agencies. These maps must be accurate and well maintained. However, in most cases, the update process of these maps is still done by hand, often using satellite or aerial imagery. Supporting this process via automatic change detection based on traditional classification algorithms is difficult due to the high level of noise in the data, such as introduced by temporary changes (e.g. cars being parked). This paper describes a method to detect changes between two time steps using 2.5D data and to transfer these insights to a digital map. For every polygon in the map, several attributes are collected from the input data, which are used to train a machine-learning model based on gradient boosting. A case study in Haarlem, in the Netherlands, was conducted to test the performance of this proposed approach. Results show that this methodology can recognize a substantial amount of changes and can support — and speed up — the manual updating process.

1. INTRODUCTION

Up-to-date digital maps are required for many applications in academia, government and industry, and they are thus produced and offered by many companies and governments worldwide. In order to keep these maps as up to date as possible, the providers of these maps face two main problems:

1. areas, especially cities, are in constant change with sometimes drastic effects; and
2. digital maps are static and can only display a snapshot of an area at a certain time.

Keeping these maps updated is vital so that they remain useful. However, this update process is still mostly done by manual workforce offered by specialized companies. Existing maps are compared with aerial images and used to change the digital map accordingly. This process is a tedious work, can take long time and some changes are not detected by the operators.

A significant step for automating this update process would be to integrate automatic change detection methods. These methods could highlight parts of the map that need further investigation and therefore speed up map updating. However, even though classical change detection algorithms for remote sensing and computer vision are able to detect differences between images, they have multiple problems. First, often it takes a substantial amount of time to compute the change detection for the complete map area. Second, these classical algorithms cannot handle well noise and temporary changes. Lastly, applying these algorithms requires advanced knowledge from many different fields of computer vision and remote sensing.

A possible solution to this problem can be the use of machine learning (ML) algorithms, which are used with great success in many problems regarding geographical data. Once the training of these algorithms is complete, they have a very high execution speed and are generally more accurate than their classical counterparts. Furthermore, only the same input data that are used for the manual updating process are needed for ML algorithms.

The goal of this paper is to find out to what extent change detection can be automated using gradient boosting, focusing on XGBoost. The research was conducted as part of a master thesis, which was done in cooperation with the company *Readar*[1]. The full thesis report is available at the online repository of TU Delft Library[2].

2. BACKGROUND

2.1 Change detection

Change detection is a common topic in remote sensing and Geographic Information Systems (GIS) and is used in many applications for academia and industry. Many different techniques have been developed for different use cases. See Ban and Yousif (2016) or Lu et al. (2004) for an overview.

In general terms, change detection can be separated in pixel-based and object-based methods (İlsever and Ünsalan, 2012; Taubenböck et al., 2012; Shi et al., 2012). The latter method is the one used in this paper, using polygons as objects.

The latest trend in change detection is the usage of ML using a variety of techniques. A rather up-to-date overview is given by Vignesh et al. (2019) and Shi et al. (2020a). An example of successful change detection using XGBoost—which is also

[1] https://readar.com/
[2] http://resolver.tudelft.nl/uuid:9a84292f-ae1c-4d39-8a8e-a0cd0c133130

used in this paper—is given by Abdullah et al. (2019), in which features from different years were processed to analyse changes in land cover.

When looking at the literature, two things can be noticed:

- In many cases the focus is set on a certain type of object, such as buildings (Matikainen et al., 2010), vegetation (Meeragandhi et al., 2015) or streets (Zhang and Couloigner, 2004). These are easier cases to handle, since the shared attributes of a single type allow for easier detection.
- Many change detection methods, especially for land cover changes, are often relying on additional data, like multispectral imagery or Radar/Lidar (Shi et al., 2020b; Kranz et al., 2017; Matikainen et al., 2010). In comparison to RGB imagery, these data are not always available or expensive to purchase.

This paper documents the attempts to overcome both of these shortcomings.

2.2 Gradient boosting

Gradient boosting is the machine-learning technique used for change detection in this paper. It combines multiple weak learners into a strong prediction model suited for both classification and regression problems. A weak learner (also called a *feature* in the context of Gradient boosting) is a single attribute of an object. In this case, an object is one polygon of the map; whereas one example of feature could be the average red value of all pixels of this specific polygon. In gradient boosting, these features are used to build *decision trees*.

A decision tree is a directed graph that depicts every possible outcome of a question/decision based on the value of certain features. A simple example of a decision tree would be a classifier which attempts to classify a set of candidates into those that like computer games and those that do not (without asking this specific question). One feature, like the gender of the candidates (with the assumption that males tend to like computer games and females tend not to), could already give a first estimation but would lead to many incorrectly classified candidates. Including more features (age, study degree, Internet usage times) can improve the results.

Boosting is a process to generate decision trees automatically from the available features using a sample of objects where the correct classification is known. Figure 1 exemplifies this process in simplified way: Trees are created iteratively to classify the values in the left plot with the available features being the x- and y-position. The trees in *gradient boosting* are created using both the features that give the best distinction between desired outcome and a random selection of features. After many iterations, many of these trees are combined into random forests and refined, with each iteration improving the result of the change detection.

Together, these random forests create a progressive model for classification or regression. In *gradient boosting*, these models are also regularized using a random differentiable loss function. An advantage of this method is its automatic handling of missing values and its fast execution time. For a more extensive explanation of gradient boosting and its uses, please refer to Natekin and Knoll (2013), Bentéjac et al. (2021) or Biau et al. (2018).

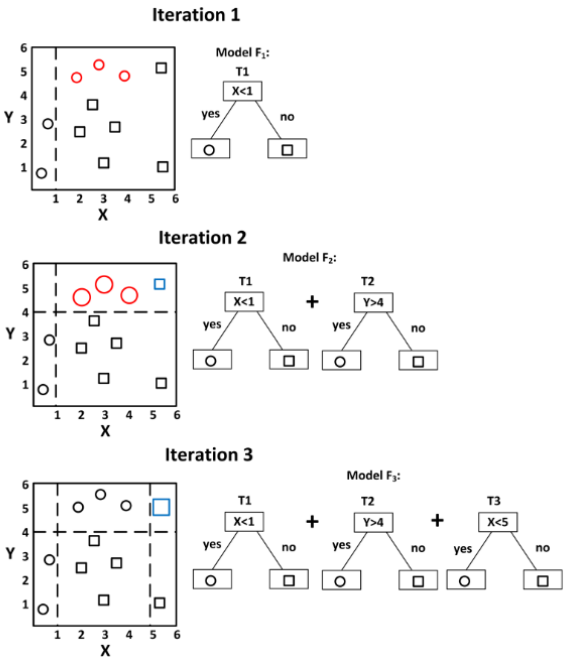

Figure 1. Functionality of Gradient boosting. © Zhang et al. (2018).

3. METHODOLOGY

This paper uses the XGBoost library developed by Chen and Guestrin (2016). It is a framework available for Python that implements gradient boosting. It has important advantages for this project, among which a very fast execution time, low resource usage and a built-in mechanism against overfitting, i.e. a situation in which the model is too closely related to the example data and cannot predict other data well.

3.1 Input data

Three different types of input data are required for change detection using our XGBoost-based approach:

- One **digital map** containing correctly categorized polygons, i.e. into buildings, waterways or vegetation. The more accurate the categorization, the better the final prediction.
- Two **aerial images**, one taken at a time that corresponds (roughly) to the information in the digital map; and one taken at the time where the changes should be detected. The higher the resolution, the more meaningful information can be extracted using the features, and the more accurate the final change detection will be.
- Two **point clouds**, also corresponding to the digital map and to the changes. They can be derived directly from aerial images using photogrammetry or based on Lidar.

3.2 What are changes?

Based on the requirements for automatic map updates and the capabilities of XGBoost, we consider a *change* as an object (i.e. polygon) that has clear differences between its two corresponding aerial images or point clouds, even if its class is the same, e.g. when an old building is replaced by a new one.

However, there are two important clarifications to be made:

- These differences must be permanent. Temporary or seasonal differences are therefore not considered as changes.
- Construction sites are considered as changes only if a change in appearance or a change in height can be found. Otherwise the training of the model is more difficult or even impossible.

3.3 Steps in methodology

The proposed approach consists of three different main processing steps, which are described in the following sections. They are:

Data preparation The input data is cleaned and prepared.

Feature extraction The features describing a change are obtained.

Training The machine learning model is created based on the features.

3.4 Data preparation

The exact preparation steps required depend on the input data, but these generally involve: accurately georeferencing the aerial images and point clouds, cleaning the digital map into a planar partition of polygons (Arroyo Ohori et al., 2012), and (manually) checking the correct classification of the polygons.

3.5 Feature extraction

An object corresponds to a polygon in the digital map, and is linked to specific subsets of the two aerial images and two point clouds via its geographical coordinates. All pixels and points that are located inside a polygon are used as the input to extract features, which are then computed for each polygon and for each time.

As described in the input data in Section 3.1, only the polygons of one time step (those in the input map, from now on referred to as *old*) are known. The polygons for the other time step (from now on referred to as *new*) would be part of the new map and are not known. Because of this, in order to get the features for old and new, the aerial images and point clouds from old and new are used, but only the polygons from old are used for the geometries (see Figure 2). All features therefore are related to the polygons from the old time step. Only if there is a change inside a polygon from the old time step, this is detected.

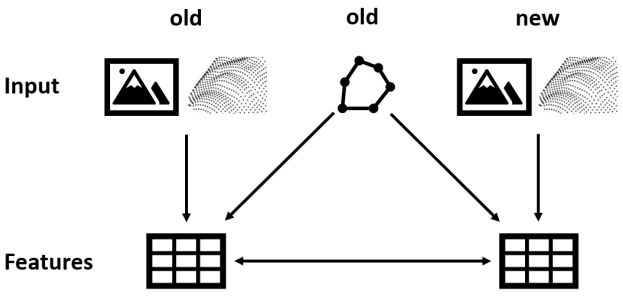

Figure 2. Components used in the feature extraction step

There are many possible features that can be extracted. This depends on the available data and the desired classification. In general, we have found that the use of four categories of features provides good results[3]:

Colour-related features Colour features describe the aerial images by means of RGB values in a polygon. They are easy to derive but very susceptible to small changes. Normal statistical functions like max, min, avg are applied to the data.

Height-features Height features are derived from the point cloud directly and contain the 3D information of the area. Next to height, aspect and slope can be created from the point cloud. Again, statistical functions are applied.

Polygon features Attributes from the polygon itself, like size or shape are used to extract features. Also the original category derived from the input map falls under this category.

Progressive features This term describes features in which the feature value is not directly derived from the input source, but must be processed first. This includes classical algorithms from computer vision, like Haralick Texture features, Fourier transform, Canny edge detection and Local binary patterns, as well as operations from GIS, such as shadow calculation.

3.6 Training

With the data from the available features, the ML-based model is trained. For this, the data from both time steps are merged, any qualitative (i.e. non-numeric) features are encoded, and the information of whether a polygon is changing between the time steps is added. For example, the latter information can be encoded as yes (1) and no (0).

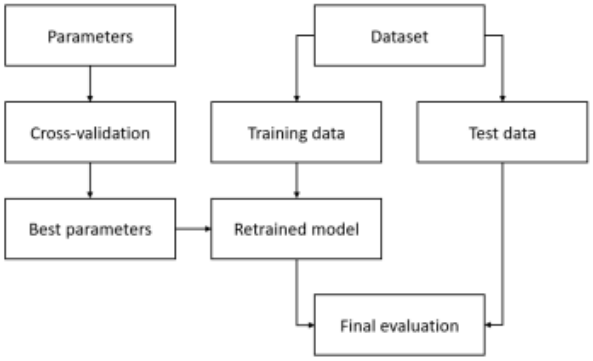

Figure 3. Workflow for tuning and training of the model.
© SciKit-Learn

The training of the model follows the best practices as can be seen in Figure 3:

1. Cross-validation is used to extract the best hyper parameters for the data;
2. The data is split into training and test sets;
3. The model is trained with the training data and the best hyper parameters;
4. The model is validated with the test set.

[3] For a more extended list of features please see the underlying MSc. thesis

Cross-validation is a process in which the whole test dataset is split in k folds. Every evaluation is done k times, each time a model is trained using $k-1$ folds and the last fold is used as validation data. This is done for different combinations of hyper parameters. Every evaluation is checked for its performance and the results of all evaluations are averaged. Afterwards, it is possible to check which set of hyper parameters gave the best results. This process is done mainly to prevent overfitting.

To make a selection between training and test sets, a percentage of the complete dataset is taken randomly. This selection is done for each category separately in order to keep the same distribution of classes in both sets.

3.7 Output and evaluation

The output of the model is a percentage value of how likely it is that a polygon changes between the two time steps.

Furthermore, the complete model must be evaluated to determine how successfully the model can classify unknown results and is therefore also a measurement of how useful the model will be for future tasks. For this evaluation, two approaches are selected.

A first evaluation is done with a *confusion matrix* (CM). The advantage of the CM is the easy visualization of the performance of the model, however the performance of the model is dependent of the threshold (the probability of change that is used to separate a change from a no-change).

Second, evaluations are done with curves that can bypass the problem of setting an initial threshold as all possible threshold values from 0 to 1 are displayed. Two curves described by Brownlee (2018) are used:

Precision-Recall curve As the name suggests, it plots the precision of a classifier with its recall. For this curve, only the correct prediction of the true target values are important. The closer the curve to the upper right corner, the better the model is at classifying the positive results correctly.

ROC Curve This curve, also known as Receiver operating characteristic (ROC)-curve, shows the trade-off between the True-Positive rate (TPR), also known as sensitivity, and False- Positive rate (FPR), also known as inverted specificity, with the former on the y-axis and the latter on the x-axis. The TPR tells what proportion of positive labelled entries are correctly labelled, the FPR tells the proportion of negative labelled entries that were incorrectly labelled. Furthermore, the AUC (area under curve) is displayed, a value describing how much area of the plot is located under the curve and which can be used to compare different curves.

4. EXPERIMENT

The methodology was implemented with the algorithms developed in Python and the visualization done in QGIS.

4.1 Study area and data

An area in the municipality of Haarlem, the Netherlands, was chosen as case study. In this area a good selection of different surface textures can be found, such as a traditional inner city core with small houses, residential areas with gardens, business and industrial areas, waterways and some agricultural areas.

Both digital map and aerial images are available for the consecutive years of 2017 and 2018. However, the digital map for 2018 will only be used for the validation of the test set.

Based on the change criteria specified in Section 3.2, a total number of 1378 polygons that are changing between 2017 and 2018 can be found. Compared to the total number of polygons, only 0.85% of the polygons are changing. Table 1 shows the exact distribution of changes. The group classification of the polygons is important for the later evaluation of the model.

Class	Group	Changes (% of class)
Building	building	464 (0.57)
Road section	street	323 (0.99)
Building installation	building	4 (0.26)
Other structures	building	4 (0.27)
Water	water	6 (0.29)
Vegetation	surface	271 (2.12)
Bare area	surface	186 (0.03)
Supporting Water	water	6 (0.2)
Supporting Road sections	street	114 (1.6)

Table 1. Changes per class

The input data for this use case has the following specifications:

- The BGT (*Basisregistratie Grootschalige Topografie*) is used as a digital map. It consists of categorized polygons in a resolution of up to 20cm.
- The aerial images are part of the *stereo10-dataset* from the company Cyclomedia. It contains contains true-ortho imagery in a resolution of 10cm. Images are taken in the leafless seasons of early spring and late autumn and with no present disturbances like clouds. A maximum visibility is therefore assured.
- The point cloud is created with 3D photogrammetry from the aerial imagery. It was created by Readar using an internally developed approach. Complementary to the default algorithms used for 3D photogrammetry, ML techniques are used to improve the results. For some parts that are occluded on the aerial images, no height information is available. However, in comparison to the total area, these parts are negligible.

4.2 Data Preparation

A grid approach, using the average height as the height of a cell, is used to create a DSM. To simplify the feature extraction, the cell size and extent of the DSM is identical to the cell size and extent of the aerial image. Afterwards QGIS is used to calculate the slope and the aspect from the DSM.

To support the manual change detection in order to have the complete dataset for training, the whole research area was tiled temporary in squares of 250×250m and each tile was checked individually for existing changes and labeled equivalently. Polygons with a surface area of less than $2m^2$ are not considered for change detection. Their size is too small to extract meaningful information and ensure a successful classification of changes.

4.3 Feature Extraction

The following features were extracted from the input data:

- Category of a polygon
- Statistical information for RGB and HSV values
- Statistical information for height, slope and aspect
- Percentage of shadow pixels in a polygon based on Bhattacharya et al. (2019)
- Haralick Texture features based on Coelho (2013)
- Local binary patterns
- Fourier transform
- Bhattacharyya distance based on Choi and Lee (2003)
- Percentages of canny edge pixels based on He et al. (2017)

The feature extraction was done by using diverse Python libraries. The following libraries were used:

NumPy	Calculating the statistical features for colour & height and calculation of Bhattacharyya distance
Pandas	Data storage and communication with XGBoost
SciPy	Extraction of computer vision features and evaluation of the model
SkiKit	Calculation of Haralick Texture features
OpenCV	Calculation of Local binary patterns, Fourier transform & Canny edge

4.4 Training

For the split between training and test sets, a ratio of 80:20 per class was chosen, resulting in 129.142 entries in the training set and 32.285 entries in the test set.

Tuning of the hyper-parameters was done with k-cross-folding, with $k = 5$. To prevent overfitting, early stopping rounds was set to 10. Table 2 describes the parameter values after tuning which delivered the best results[4]:

Parameter	Initial value
objective	binary:logistic
eval_metric	aucpr
learning_rate	0.3
max_depth	7
min_child_weight	3
gamma	0.0
colsample_bytree	0.8
subsample	0.9
scale_pos_weight	10.78
reg_alpha	1e-5
n_estimators	394

Table 2. Overview of tuning parameters with initial values

Tweaking the parameters only results in small changes in the end result. However, two parameters are important for a successful change detection. The maximum depth of a decision tree should be at least seven, as otherwise the complexity of the model is too low to distinguish between temporary and permanent changes. Furthermore, it is very important that a higher weight is given to positive examples (i.e. changes) (with scale_pos_weight). Otherwise, the majority of polygons, which likely have no change, can influence the results (e.g. by classifying everything as not changed).

[4] For an exact description of the parameters, the reader is referred to `https://xgboost.readthedocs.io/en/latest/parameter.html`

The actual training of the model is simple: an XGBoost model with the tuned parameters is created as a Python object. Afterwards, the training data (both the features and the target values) are given to the model and the training can start.

5. RESULTS

Figure 4 shows how the outcome of the model could be presented to a person responsible for updating the map. The more blue a polygon is, the more likely a change has occurred between the two time steps. Instead of considering the whole area, the focus can be set directly on interesting areas. Note that in this image, only the subset of polygons that are part of the test set are shown.

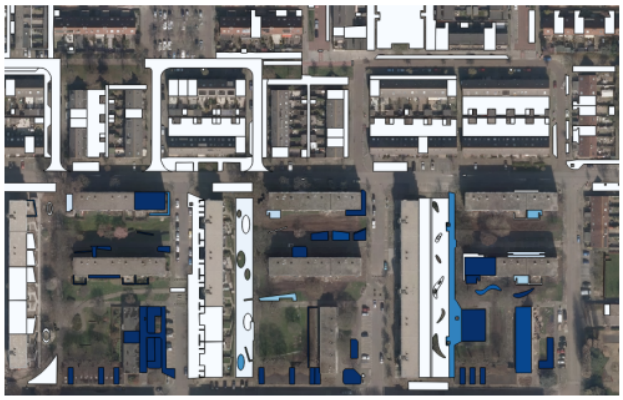

Figure 4. Example view for the operator

Figure 5 shows both examples for a detected and for an undetected change detection. The left images are from the old time step, the right images are from the new time step. In both cases, a change has occurred between both time steps. The reason for the detection failure is that the difference between both images is too small: a patch of sand in the old time step was replaced by concrete with some sandy parts in the new time step. This results in very similar feature attributes and thus the change fails to be automatically recognized.

In the following sections, the curves and confusion matrix of the results of the case study are shown. The evaluations were created using the Python library SKLearn.

5.1 Confusion matrices

As can be seen in Table 3, the results are very dependent on the threshold which is used to separate changes from no changes.

Threshold	TN	FP	FN	TP
90%	32 006	3	160	116
50%	31 996	13	128	148
10%	31 931	78	89	187
1%	31 600	409	63	213
0.1%	30 248	1761	22	254

Table 3. Results for confusion matrix based on different thresholds

For some object types, changes can be detected more easily than for others. To highlight this, Table 4 shows the CM for different BGT-groups. As the highest number of TP were found with the threshold of 0.001, the threshold for these matrices is likewise set to this value.

Figure 5. Examples of successful (top) and unsuccessful (bottom) change detection

BGT-group	TN	FP	FN	TP
building	16 478	215	5	90
street	6932	585	8	80
surface	5846	585	8	83
water	992	17	1	1

Table 4. Results for confusion matrix based on different groups

5.2 Curves

Figure 6 displays the PR-curve with the precision and recall for different thresholds in orange. The blue line shows the precision of a classifier with no skill and is related to the percentage of changes in the dataset. In this model the highest threshold is on the left and the lowest on the right.

- Up to a threshold of 60%, the precision is close to 1, meaning almost no FP are detected but there are a large number of FN.
- From a threshold of 60% to a threshold to 40%, the precision stays relatively flat, that means while gaining a smaller amount of FN, the number of FP is only rising slowly.
- From a threshold of 40% to 20%, the precision is falling more than the recall is rising, hence the number of detected FP is rising more quickly than the number of detected TP.
- From a threshold of 20% to 0%, the numbers of FP and TP are balanced.

The second curve (Figure 7) displays the ROC-Curve with the inverted FPR (x-axis) and the TPR (y-axis) for different thresholds. The blue line represents the condition where TPR equals FPR. Going right on the x-axis means to lower the threshold and find more FP. The following can be noted for this curve:

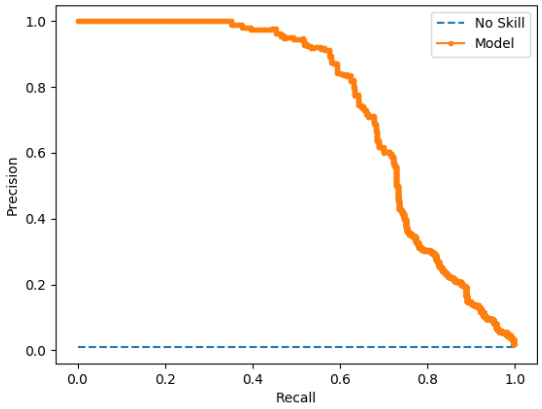

Figure 6. Precision-Recall curve

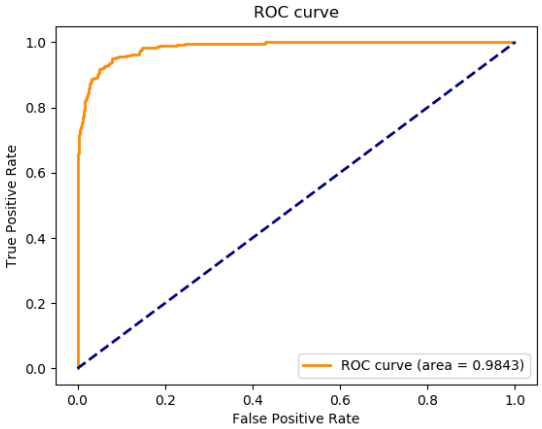

Figure 7. ROC curve

- At an FPR around 0, the line rises quickly to a TPR of 0.7, that means 70% of the changes can be identified with very few FP.
- Until an FPR of 0.05 the curve is very steep, so with a small increase in the number of FP many TP can be found.
- From an FPR of 0.05 to 0.4 the curve is rising less, so small increases in the number of FP only give small increases in the number of TP.
- Over an FPR of 0.4 the number of FP is rapidly increasing while the number TP is almost not increasing.

5.3 Baseline

A common approach in ML is to apply simple algorithms to the data in order to have a point of comparison to the more advanced algorithms. These algorithms are referred to as baseline predictions. For this case study, logistic regression and isolation forest (usually taken as outlier detection and used here as there is only a very small number of changes) are taken as baselines.

Type	Acc.	Prec.	Recall	F1-Score	ROC-AUC	PR-AUC
Log. Reg.	0.651	0.0198	0.8225	0.0387	0.8199	0.0783
Iso. For	0.8497	0.0252	0.3007	0.0466	0.6003	0.0146

Table 5. Results for baseline models

It can be seen that the algorithm developed and presented here outperforms the baseline predictions across all evaluation criteria.

6. CONCLUSIONS

In our case study, when distinguishing between changes and non-changes, the method can determine with absolute certainty that a change has not occurred for a majority of cases. For 99% of the entries the probability of it being a change is below 1%. On the other hand, when classifying changes, the model is less clear as represented by a higher spread of probabilities for changes. Approximately the same number of changes have a high probability of being a change as the same number of changes having a low probability of being a change. An explanation for this observation could be the different amount of training data available for both classes. For non-changes, much more training data is available and the model can better learn which details matter. For changes on the other hand, less training data is available. It is sufficient to establish a detection, however not enough to recognize the details for every kind of change.

When looking at the CM divided in different classes, it can be noticed that changes concerning buildings can be distinguished better. Even though there are slightly more FN (caused by not detecting changes in buildings in which only the surface changed, for example new solar panels) the number of FP is lower. That could be explained by the fact that in this group the influence of height differences are particularly relevant. Streets and surfaces on the other hand have a higher number of FP. Temporary changes can mostly be found in this group, leading to the high numbers. Change detection for waterways tends to be the most difficult. However, considering the fact that changes of areas of water are rare, this problem is deemed less important.

The curves show that the model is very certain about classifying around 50% of the changes, which have a very high probability of being a change (left part of PR-Curve). The precision is very high, but as only half of the changes are classified, the recall is not very good. To get almost all changes (recall over 95%), the precision is going down to 10%, that means 90% of all classified changes are FP. However, considering the small number of recognized changes (true and false) compared to the total number of polygons, even the classification with a low precision can be accepted as helpful.

It can be emphasized that change detection using XGBoost is possible and can be semi-automated to support the manual change detection. As can be seen in Table 3, depending on the threshold, a large number of changes can be found. With the recommended threshold of 0.01, around 80% of the changes can be found by checking only 5% of all polygons manually.

Under the assumption that change detection is still done manually in a Geographic information system (GIS) by comparing aerial images, and the assumption that polygons in close distance to the TP are also in the range of visual perception, the results look even more promising. Around 95% of all changes are located in a maximum distance of 10m around the polygons classified as TP.

However, the biggest challenge is detecting polygons that change without an accompanying change in height. These are difficult to recognize and cannot be found at close distance to other changes. To detect these changes, manual checking is still required. In conclusion, this algorithm is suitable for the pre-selection of polygons in which a change is more likely and should be used for this. It is not recommended to rely on it as the only change detection tool.

Regarding the evaluation results, currently the verification of the model is done with a small subset of the same dataset. Even though many measures are taken against overfitting and the model works properly, further tests on different datasets are needed to ensure that the model is working properly.

6.1 Recommendations

If the data is likely to have errors (for example spikes in the point cloud data), it is better to use percentiles instead of absolute values as features. Instead of describing the values of outliers, this is then more likely to describe the real characteristics of the data.

An easy way to get more features with a better model for change detection is to not only use the features, but also calculate the difference between features (avg height for time step 1, avg height for time step 2, difference between time step 1 and time step 2). This allows a more direct inclusion of the information of how big the change between two features is, instead of deducing it from the two values, which results in a better model.

6.2 Future Work

The quality of ML algorithms greatly depends on the data available. Examples are needed to derive rules and improve the results. More training data can therefore improve the results. Having more data means that more different situations in which changes are occurring can be considered. An increased number of positive results (changes) would be especially helpful, as most datasets are very imbalanced with only a fraction being changes.

An integration of deep learning methods could help improve the quality. It could either be used to get many new features as an additional input for an XGBoost-related algorithm (for example with ResNet) and improve the results of the XGBoost-based change detection. Another approach would be to replace the complete XGBoost change detection with deep learning as done by de Jong and Bosman (2019) with a change detection for satellite images. The network should then be able to learn autonomously which attributes of the input data are important and learn to classify changes. However, in comparison to XGBoost, this approach needs advanced GPU-based machines, a lot more processing/debugging time and it would be a complete black box to the user (in the sense that neither the features nor their importance are known).

ACKNOWLEDGEMENTS

We thank Readar for providing both the data and the opportunity to work on this problem. We thank Matthijs van Til for its feedback for the result presentation and Jean-Micheal Renders for the good hints while building the Model. Furthermore we would like to thank Roderik Lindenbergh for his comments on the manuscript. We would like to thank Bert Wouters for his support in the last stage of this paper.

This study has received funding from the European Research Council (ERC) under the European Union's Horizon2020 Research & Innovation Programme (grant agreement no. 677312 Urban modelling in higher dimensions).

REFERENCES

Abdullah, A. Y. M., Masrur, A., Adnan, M. S. G., Baky, M. A. A., Hassan, Q. K., Dewan, A., 2019. Spatio-temporal Patterns of Land Use/Land Cover Change in the Heterogeneous Coastal Region of Bangladesh between 1990 and 2017. *Remote Sensing*, 11(7), 790.

Arroyo Ohori, K., Ledoux, H., Meijers, M., 2012. Validation and Automatic Repair of Planar Partitions Using a Constrained Triangulation. *Photogrammetrie, Fernerkundung, Geoinformation*, 5, 613–630. ISSN: 1432–8364.

Ban, Y., Yousif, O., 2016. Change Detection Techniques: A Review. Y. Ban (ed.), *Multitemporal Remote Sensing*, 20, Springer International Publishing, Cham, 19–43.

Bentéjac, C., Csörgő, A., Martínez-Muñoz, G., 2021. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3), 1937–1967. http://link.springer.com/10.1007/s10462-020-09896-5.

Bhattacharya, S., Braun, C., Leopold, U., 2019. A Novel 2.5D Shadow Calculation Algorithm for Urban Environment:. *Proceedings of the 5th International Conference on Geographical Information Systems Theory, Applications and Management*, SCITEPRESS - Science and Technology Publications, 274–281.

Biau, G., Cadre, B., Rouvière, L., 2018. Accelerated Gradient Boosting. https://arxiv.org/abs/1803.02042v1.

Brownlee, J., 2018. How to Use ROC Curves and Precision-Recall Curves for Classification in Python. Available at `https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python`.

Chen, T., Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 785–794.

Choi, E., Lee, C., 2003. Feature extraction based on the Bhattacharyya distance. *Pattern Recognition*, 36(8), 1703–1709.

Coelho, 2013. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, 1(1), e3.

de Jong, K. L., Bosman, A. S., 2019. Unsupervised Change Detection in Satellite Images Using Convolutional Neural Networks. *arXiv:1812.05815 [cs]*. http://arxiv.org/abs/1812.05815.

He, A., He, J., Kim, R., Like, D., Yan, A., 2017. An ensemble-based approach for classification of high-resolution satellite imagery of the Amazon Basin. *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, IEEE, Cambridge, MA, 1–4.

İlsever, M., Ünsalan, C., 2012. *Two-dimensional change detection methods: remote sensing applications*. SpringerBriefs in computer science, Springer, London.

Kranz, O., Lang, S., Schoepfer, E., 2017. 2.5D change detection from satellite imagery to monitor small-scale mining activities in the Democratic Republic of the Congo. *International Journal of Applied Earth Observation and Geoinformation*, 61, 81–91.

Lu, D., Mausel, P., Brondízio, E., Moran, E., 2004. Change detection techniques. *International Journal of Remote Sensing*, 25(12), 2365–2401.

Matikainen, L., Hyyppä, J., Ahokas, E., Markelin, L., Kaartinen, H., 2010. Automatic Detection of Buildings and Changes in Buildings for Updating of Maps. *Remote Sensing*, 2.

Meeragandhi, G., Arun, S., Thummala, N., Christy, A., 2015. Ndvi: Vegetation Change Detection Using Remote Sensing and Gis – A Case Study of Vellore District. *Procedia Computer Science*, 57, 1199-1210.

Natekin, A., Knoll, A., 2013. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7.

Shi, J., Wang, J., Xu, Y., 2012. Object-based change detection using georeferenced UAV images. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-1/C22, 177–182.

Shi, W., Zhang, M., Zhang, R., Chen, S., Zhan, Z., 2020a. Change Detection Based on Artificial Intelligence: State-of-the-Art and Challenges. *Remote Sensing*, 12(10).

Shi, W., Zhang, M., Zhang, R., Chen, S., Zhan, Z., 2020b. Change Detection Based on Artificial Intelligence: State-of-the-Art and Challenges. *Remote Sensing*, 12(10), 1688. https://www.mdpi.com/2072-4292/12/10/1688.

Taubenböck, H., Esch, T., Felbier, A., Wiesner, M., Roth, A., Dech, S., 2012. Monitoring urbanization in mega cities from space. *Remote Sensing of Environment*, 117, 162–176.

Vignesh, Thyagharajan, Ramya, 2019. Change Detection using Deep Learning and Machine Learning Techniques for Multispectral Satellite Images. *International Journal of Innovative Technology and Exploring Engineering*, 9(1S), 90–93.

Zhang, Q., Couloigner, I., 2004. Automatic road change detection and GIS updating from high spatial remotely-sensed imagery. *Geo-spatial Information Science*, 7, 89-95.

Zhang, Z., Mayer, G., Dauvilliers, Y., Plazzi, G., Pizza, F., Fronczek, R., Santamaria, J., Partinen, M., Overeem, S., Peraita-Adrados, M., Silva, A., Sonka, K., Río, R., Heinzer, R., Wierzbicka, A., Young, P., Högl, B., Bassetti, C., Manconi, M., Khatami, R., 2018. Exploring the clinical features of narcolepsy type 1 versus narcolepsy type 2 from European Narcolepsy Network database with machine learning. *Scientific Reports*, 8.