

An aerial photograph of a city street grid with a river. The image is overlaid with a semi-transparent white box containing text. The map features red and green color-coded areas, likely representing different types of buildings or land use. The river is blue, and the streets are grey.

# Automatic change detection in digital maps using aerial images and point clouds

Felix Dahle

June 2020



# AUTOMATIC CHANGE DETECTION IN DIGITAL MAPS USING AERIAL IMAGES AND POINT CLOUDS

A thesis submitted to the Delft University of Technology in partial fulfillment  
of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Felix Dahle

June 2020

Felix Dahle: *Automatic change detection in digital maps using aerial images and point clouds* (2020)

© This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The work in this thesis was carried out in the:



3D geoinformation group  
Department of Urbanism  
Faculty of the Built Environment & Architecture  
Delft University of Technology

Supervisors: Dr. Ken Arroyo Ohori

Dr. Giorgio Aguiaro

Co-reader: Dr. Roderik Lindenbergh

## ABSTRACT

In many countries digital maps are created and provided by the national cadastres: Usually they consist of multiple polygons, each with an exact location and shape, describing which kind of surface can be found at the position of the polygon (e. g. building, street, vegetation). They must be accurate and well maintained, as they are used by companies or authorities for purposes like urban planning or demographic statistics. However, especially cities are in a constant change. Old buildings are torn down, new buildings are built, and complete streets and neighbourhoods are changed. Monitoring these changes is difficult and identifying and updating the virtual maps is still done mostly manually today.

A method is developed to detect changes on the ground and identify changes for the virtual maps automatically using machine learning approaches. As input data the virtual map, their corresponding aerial images and point clouds from different years are needed. As a case study for this thesis, this method is developed and applied to the BGT, the Dutch virtual map with a resolution of 20cm. The research area is the city of Haarlem for 2017 and 2018. High resolution aerial images are used in combination with point clouds created by Photogrammetry.

The output is again a digital map of the area where every polygon has a probability score of how likely its category (for example building, street, etc..) changed. This can support the manual updating process eminently, as a minor percentage of polygons (for which the algorithm was unsure) must be checked manually. The research question of this thesis is to check whether this change detection is feasible even for highly heterogeneous structures like cities. Many visual changes in the aerial images are happening that are not relevant for the virtual map. In the one year, a street can be full of colourful cars, in the other year, the street is empty and completely grey. Many scenes are easy for humans to distinguish but are challenging for an algorithm. The goal is to detect a high amount of true changes while keeping the number of false positives low to reduce the manual work as much as possible.

To achieve good change detection and answer the research questions, the machine learning library of XGBoost is used. It provides a gradient boosting framework for many different environments, including Python. Many weak learners, each classifying a change only with a very low detection rate (for example minimum height of all points inside the polygon), are combined to get a strong learner. This learner should be able to classify polygons with a high accuracy into polygons that change and polygons that do not change.

With this method it is possible to detect a high amount of changes. 80% of all changes can be found within a reasonable number of False positives. Especially for buildings almost all changes can be identified. It is furthermore possible to localize the changes in larger polygons. However, not all changes can be identified, so that this approach should be seen as an aid for manual change detection and not to replace it.

## ACKNOWLEDGMENTS

First of all, I would like to thank my supervisors Ken Ohori and Giorgio Aguiaro for both their great feedback on my thesis and their guidance through the jungle of rules and the required administrative work.

Furthermore I would like to thank Readar for giving me the opportunity to work on this project and refine my skills. Especially should be mentioned Sven Briels, Jean-Micheal Renders and Matthijs van til for their amazing suggestions during the project. Also I like to thank my Co-Reader Roderik Lindenbergh for his valuable feedback.

Finally a big thank you for Anja Unger and Christian Dahle for reading the Thesis extensively and helping me to find annoying errors and to create this document.

# CONTENTS

|       |                               |    |
|-------|-------------------------------|----|
| 1     | INTRODUCTION                  | 1  |
| 1.1   | Motivation                    | 2  |
| 1.2   | Research Questions            | 3  |
| 1.3   | Scope                         | 4  |
| 1.4   | Thesis Outline                | 4  |
| 2     | THEORETICAL BACKGROUND        | 5  |
| 2.1   | Input data                    | 5  |
| 2.1.1 | BGT                           | 5  |
| 2.1.2 | Aerial images                 | 6  |
| 2.1.3 | Point cloud                   | 7  |
| 2.1.4 | DSM                           | 9  |
| 2.2   | Machine Learning              | 9  |
| 2.2.1 | Learning methods              | 10 |
| 2.2.2 | Categories                    | 10 |
| 2.3   | Gradient Boosting             | 11 |
| 2.3.1 | Decision Trees                | 11 |
| 2.3.2 | Functionality                 | 13 |
| 2.3.3 | XGBoost                       | 15 |
| 2.4   | Features                      | 18 |
| 2.4.1 | Colour features               | 18 |
| 2.4.2 | Height features               | 19 |
| 2.4.3 | Polygon features              | 19 |
| 2.4.4 | Progressive features          | 19 |
| 3     | RELATED WORK                  | 22 |
| 3.1   | Traditional methods           | 23 |
| 3.2   | Height methods                | 24 |
| 3.3   | Machine learning methods      | 25 |
| 3.4   | Conclusion                    | 26 |
| 4     | METHODOLOGY                   | 28 |
| 4.1   | Workflow                      | 28 |
| 4.2   | Preparation of the data       | 30 |
| 4.2.1 | Convert point cloud           | 30 |
| 4.2.2 | Clean polygons                | 30 |
| 4.2.3 | Identify changes              | 31 |
| 4.3   | Feature extraction            | 32 |
| 4.3.1 | Clipping Input data           | 33 |
| 4.3.2 | Calculating features          | 34 |
| 4.3.3 | Merging data                  | 34 |
| 4.3.4 | Preparation of the features   | 35 |
| 4.4   | Model training                | 35 |
| 4.4.1 | Splitting the data            | 36 |
| 4.4.2 | Model parameters              | 37 |
| 4.4.3 | Parameter tuning              | 38 |
| 4.4.4 | Training & applying the model | 39 |
| 4.5   | Model evaluation              | 39 |
| 4.5.1 | Feature evaluation            | 40 |
| 4.5.2 | Result evaluation             | 41 |
| 4.6   | Locating changes in polygons  | 44 |
| 4.7   | Possible Obstacles            | 44 |

|       |  |    |
|-------|--|----|
| 5     | IMPLEMENTATION                                       | 46 |
| 5.1   | Research area  | 46 |
| 5.2   | Input data   | 46 |
| 5.2.1 | BGT  | 46 |
| 5.2.2 | Aerial images  | 48 |
| 5.2.3 | Point cloud  | 48 |
| 5.2.4 | Changes  | 48 |
| 5.3   | Tools  | 48 |
| 5.3.1 | PostgreSQL   | 49 |
| 5.3.2 | Python   | 49 |
| 5.3.3 | XGBoost  | 50 |
| 5.3.4 | Matlab   | 50 |
| 5.3.5 | QGIS   | 50 |
| 5.4   | Design decisions                                     | 50 |
| 5.4.1 | Data preparation                                     | 51 |
| 5.4.2 | Feature extraction                                   | 51 |
| 5.4.3 | Model training                                       | 53 |
| 5.4.4 | Model evaluation                                     | 54 |
| 5.4.5 | Locating changes                                     | 54 |
| 5.5   | Computation times                                    | 55 |
| 6     | RESULTS  | 56 |
| 6.1   | Model results  | 56 |
| 6.1.1 | Confusion matrices                                   | 57 |
| 6.1.2 | Scores   | 57 |
| 6.1.3 | Curves   | 58 |
| 6.2   | Feature importance                                   | 61 |
| 6.3   | localizing changes                                   | 62 |
| 6.4   | Comparisons  | 63 |
| 6.4.1 | Comparison to BGT changes                            | 63 |
| 6.4.2 | Comparison to baseline predictions                   | 64 |
| 6.5   | Recommendations                                      | 64 |
| 7     | DISCUSSION   | 66 |
| 7.1   | Final Remarks  | 66 |
| 7.1.1 | Feature importance                                   | 66 |
| 7.1.2 | Model results  | 67 |
| 7.2   | Common classification problems                       | 68 |
| 7.3   | Research questions                                   | 69 |
| 7.4   | Contributions  | 71 |
| 7.5   | Future Work  | 72 |
| 7.5.1 | More training data                                   | 72 |
| 7.5.2 | Deep learning  | 72 |
| 7.5.3 | Creation of a public test-set                        | 73 |
| 7.6   | Reflection   | 73 |
| 8     | ANNEX  | 81 |
| 8.1   | Example for features                                 | 81 |
| 8.2   | Results for unsuccessful attempts                    | 82 |
| 8.3   | Examples for successful change detection             | 83 |
| 8.4   | Results for the different feature importance options | 84 |
| 9     | HYPERLINKS   | 85 |



# LIST OF FIGURES

|             |  |    |
|-------------|--|----|
| Figure 1.1  | Digital map . . . . .                              | 1  |
| Figure 1.2  | Incorrect digital map . . . . .                    | 2  |
| Figure 2.1  | Digital map and aerial image . . . . .             | 6  |
| Figure 2.2  | Ortho and trueortho . . . . .                      | 7  |
| Figure 2.3  | Underfitting and Overfitting . . . . .             | 10 |
| Figure 2.4  | Decision Tree . . . . .                            | 12 |
| Figure 2.5  | Decision Tree II . . . . .                         | 14 |
| Figure 2.6  | Gradient boosting model creation . . . . .         | 15 |
| Figure 2.7  | Evolution of tree-based decision systems . . . . . | 16 |
| Figure 2.8  | Decision Tree XGBoost . . . . .                    | 17 |
| Figure 2.9  | Decision Tree XGBoost II . . . . .                 | 17 |
| Figure 2.10 | Haralick features . . . . .                        | 19 |
| Figure 2.11 | Fourier transform . . . . .                        | 20 |
| Figure 2.12 | Local binary pattern . . . . .                     | 20 |
| Figure 3.1  | Overview papers . . . . .                          | 26 |
| Figure 4.1  | Workflow . . . . .                                 | 29 |
| Figure 4.2  | Iterative Workflow . . . . .                       | 30 |
| Figure 4.3  | Overlapping Polygons . . . . .                     | 31 |
| Figure 4.4  | Feature extraction . . . . .                       | 33 |
| Figure 4.5  | Non rectangular Polygon . . . . .                  | 33 |
| Figure 4.6  | Categorical encoding . . . . .                     | 35 |
| Figure 4.7  | Tuning and Training of XGBoost . . . . .           | 36 |
| Figure 4.8  | K-cross-folding . . . . .                          | 36 |
| Figure 4.9  | Temporal differences in aerial images . . . . .    | 45 |
| Figure 5.1  | Location of Haarlem . . . . .                      | 46 |
| Figure 5.2  | Overview BGT classes . . . . .                     | 47 |
| Figure 5.3  | Database scheme for the features . . . . .         | 49 |
| Figure 6.1  | Example change detection results . . . . .         | 56 |
| Figure 6.2  | Evaluation scores . . . . .                        | 58 |
| Figure 6.3  | ROC-Curve . . . . .                                | 59 |
| Figure 6.4  | PR-Curve . . . . .                                 | 60 |
| Figure 6.5  | Economic curve . . . . .                           | 60 |
| Figure 6.6  | Feature Importance SKlearn . . . . .               | 61 |
| Figure 6.7  | Feature Importance SHAP . . . . .                  | 62 |
| Figure 6.8  | Localized changes . . . . .                        | 63 |
| Figure 6.9  | Spikes in DSM . . . . .                            | 65 |
| Figure 7.1  | Changing polygons . . . . .                        | 68 |
| Figure 8.1  | Successful change detection . . . . .              | 83 |
| Figure 8.2  | Feature Importance SKlearn (Large) . . . . .       | 84 |

## LIST OF TABLES

|            |  |    |
|------------|--|----|
| Table 2.1  | Overview machine learning tasks . . . . .    | 11 |
| Table 2.2  | Gradient Boosting dataset . . . . .          | 13 |
| Table 2.3  | Gradient Boosting dataset II . . . . .       | 13 |
| Table 2.4  | Gradient boosting dataset III . . . . .      | 15 |
| Table 2.5  | XGBoost Dataset . . . . .                    | 16 |
| Table 3.1  | Overview Papers . . . . .                    | 26 |
| Table 4.1  | Example for Confusion matrix . . . . .       | 41 |
| Table 5.1  | Numbers for BGT-objects . . . . .            | 47 |
| Table 5.2  | Class changes . . . . .                      | 48 |
| Table 5.3  | Tuning parameters . . . . .                  | 53 |
| Table 6.1  | Results confusion matrix threshold . . . . . | 57 |
| Table 6.2  | Results confusion matrix group . . . . .     | 57 |
| Table 6.3  | Results confusion matrix adapted . . . . .   | 57 |
| Table 6.4  | Score - Accuracy . . . . .                   | 57 |
| Table 6.5  | Score - Precision . . . . .                  | 58 |
| Table 6.6  | Score - Recall . . . . .                     | 58 |
| Table 6.7  | Score - F <sub>1</sub> . . . . .             | 58 |
| Table 6.8  | Evaluation for feature importance . . . . .  | 62 |
| Table 6.9  | Evaluation scores split polygons . . . . .   | 62 |
| Table 6.10 | Confusion matrix split polygons . . . . .    | 63 |
| Table 6.11 | Results for logistic regression . . . . .    | 64 |
| Table 6.12 | Results for Outlier detection . . . . .      | 64 |
| Table 7.1  | Important features . . . . .                 | 70 |

# ACRONYMS

|            |  |    |
|------------|--|----|
| <b>AUC</b> | Area under curve                         | 44 |
| <b>BGT</b> | Basisregistratie Grootchalige Topografie | 2  |
| <b>CM</b>  | Confusion Matrix                         | 41 |
| <b>CNN</b> | Convolutional Neural Network             | 25 |
| <b>DSM</b> | Digital Surface Model                    | 5  |
| <b>FN</b>  | False Negative                           | 41 |
| <b>FP</b>  | False Positive                           | 41 |
| <b>FPR</b> | False Positive rate                      | 42 |
| <b>GIS</b> | Geographic information system            | 69 |
| <b>HSV</b> | Hue Saturation Value                     | 18 |
| <b>LBP</b> | Local Binary Patterns                    | 20 |
| <b>ML</b>  | Machine Learning                         | 2  |
| <b>NaN</b> | Not a Number                             | 30 |
| <b>PR</b>  | Precision-Recall                         | 42 |
| <b>QoL</b> | Quality of life                          | 18 |
| <b>RGB</b> | Red Green Blue                           | 18 |
| <b>ROC</b> | Receiver operating characteristic        | 42 |
| <b>TN</b>  | True Negative                            | 41 |
| <b>TNR</b> | True negative rate                       | 42 |
| <b>TP</b>  | True Positive                            | 41 |
| <b>TPR</b> | True positive rate                       | 42 |



# 1

## INTRODUCTION

Many countries and companies offer freely available digital maps that can be used by everybody and that play an important role as valuable input data for projects of many stakeholders, let it be companies, scientist or authorities. According to the study of the consulting company [AlphaBeta \[2017\]](#), digital maps generate customer benefits worth over 550 billion US\$ and create approximately 4 million jobs. These digital maps usually consist of polygons that describe the exact shape, surface type and depending on the map, additional information like the construction date for buildings or vegetation type. Figure 1.1 shows an exemplary digital map of Amsterdam.

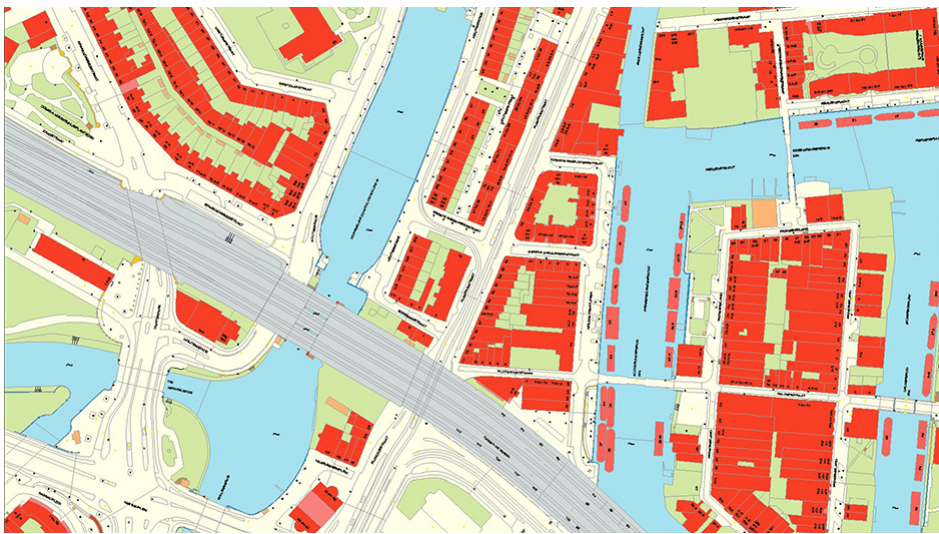


Figure 1.1: Example for a digital map (© Gemeente Amsterdam)

Providers of these digital maps are facing two problems:

1. Cities are the subject of constant change which sometimes changes having an influence to the whole structure of neighbourhoods. For example, according to the statistic authority of the Netherlands in 2018 more than 71.000 new buildings (around 6%) were built in the Netherlands<sup>1</sup>. But not only buildings, also streets and public places are changing regularly.
2. Digital maps on the other hand are static and can only display a snapshot of the structure of a city at a certain time. Static in this context means that once the map is created and uploaded to a server, it will remain in this version for a certain time. Usually, the accuracy of the digital map is sufficient for a while, but inevitably these maps must be updated again.

Hence, in order to be useful, these maps must be constantly updated and all changes on the surface must be considered and applied. This updating process is still mostly done by a manual workforce of specialized companies: Employees are considering the existing map and aerial imagery from both the date of the map and the latest date of recording. If they recognize changes between the images, the map must be changed accordingly.

<sup>1</sup> see [CBS](#).

The goal of this thesis is to support the updating process by an automatic change detection with a Machine Learning (ML)-algorithm. These kind of algorithms are used with great success in many problems regarding geographical data, so that it seemed to be a promising approach to consider for this thesis. Due to their efficiency and popularity among the ML-community, the library of XGBoost was selected for this project. More reasons can be found in § 2.3.3.

Aerial images and point clouds of two different years are incorporated with existing maps as the source for polygons. For every polygon for both years different information (features) are extracted. Based on these features, the algorithm should be able to detect changes between both years. Each polygon of the digital map will be given a probability of how likely a change has happened between the two time steps. As a large number of polygons will be processed, a high degree of automatization and efficiency is desired. However, this detection is not a trivial task, as there are many obstacles for change detection, e.g. temporary changes or heterogeneous classes. These obstacles must be considered and algorithms developed that can still enable a reasonably accurate detection of true changes.

In this thesis the whole approach is executed as a use-case using data from the Basisregistratie Grootschalige Topografie (BGT), the digital map of the Netherlands from the Dutch government. It is already existing for multiple years so that different versions of the same area are available for two consecutive years. Furthermore, suitable input data (aerial images & point clouds) in high quality are at hand for both respective years. However, this algorithm is not limited to this specific input data, but can be used for every digital map with categorized polygons.

## 1.1 MOTIVATION

The problem described in the introduction is addressed by Readar<sup>2</sup>, the company in which the research is carried out. The company is based in the Netherlands and specialized in data mining from aerial images and height data. Many municipalities of the Netherlands are already customers of the company, so that the current demand for an automatic change detection is known.

The manual updating process described previously is prone to errors: Changes can be very small, e.g. a small new path is built in a park and partly hidden under trees and can be overlooked, or actual changes can be handled incorrectly as can be seen in Figure 1.2, where the house in the middle of the figure is new and the polygon was not drawn accordingly. With the change detection developed in this thesis, these errors cannot be prevented, but will be recognized during the change detection so that they can be corrected.



Figure 1.2: polygon of a house (red) not correctly drawn

<sup>2</sup> for more information see [Readar](#).

Moreover, the opposite is also possible, that changes are plotted even though they do not exist, for example a truck standing close to a building is recognized as an extension of the building. Furthermore, the classification of when a change happened can be very subjective, e.g. if a construction site is already a building or still bare terrain. The results of this manual change detection are fluctuating quality of the updated maps.

Another big drawback of the manual approach is the time needed for updating. As complete municipalities must be considered, the area that must be checked manually can be quite large. Even though it seems to be a promising strategy for manual change detection to only focus on completely new built areas in which many changes are happening, changes can also consist just for one house in an otherwise untouched area. In every case, the whole area must be searched.

Developing a new supporting algorithm can both support the quality and decrease the time needed for change detection. After applying this algorithm, every polygon will have a probability-score describing how likely this polygon has changed. Focus can be set on the polygons with a higher probability instead of having to check the whole area for changes. Depending on the minimum probability until which polygons are checked, a certain percentage of all changes can be guaranteed to be recognized.

Instead of using traditional pixel- or object based comparison methods, the focus will be set on ML, more precisely gradient boosting using XGBoost. An explanation of this library can be found in § 2.3.3.

## 1.2 RESEARCH QUESTIONS

This study will investigate if ML using gradient boosting methods like XGBoost is able to support change detection for digital maps. The main research question for this thesis is:

*To what extent can the change detection be automatized using Machine Learning algorithms like gradient boosting?*

Automatizing the whole process of recognizing changes is a challenging task. Even with good input data the same surface or structures can be diverse in texture. Vegetation in spring is mostly green, whereas in late autumn the trees usually have no leaves and therefore less green pixels. Streets can be grey, but also multicoloured if the aerial image is taken while many cars were waiting on this street during a traffic jam. Still these classes must be recognized as identical and labelled as no change. This thesis researches to what extent this can be done by ML-algorithms and how good the results describe the reality.

Additionally to the main research question the following sub-questions will be answered:

1. *Which features of the input data can be used in terms of costs and benefits?*

Even though it would be possible to just collect as many features as possible this would not be a good approach. Some features will be very important whereas other features will hardly improve the results. Less features means less preparation of the data and less processing time in total. Thus, it must be assessed which features are beneficial for the change detection and still reasonable in the effort to get them.

2. *Which information except height can be used from 3D point clouds?*

Incorporating point clouds next to the aerial images allows to use a lot of new features that can be used for an extended change detection. Some features from 3D point clouds are easy to get and used in many applications, namely statistical information (min, max, avg, ..) for the height. This thesis will examine which additional features can be derived from the point clouds.

3. *Which metric can be used to evaluate the results?*

A reasonable evaluation of the results is very important and there are many metrics to determine the success of the ML-model. But not every metric can be used equally well. The metrics that can be used to evaluate the results will be investigated.

4. *Is it possible to locate the exact position in which the change has happened?*

For some polygons a change does not affect the complete polygon, but only a small subset of it. A good example would be large field in which a small house has been built. Especially for large polygons it would be helpful to locate exactly where the change in the polygon happened and therefore increase the usability of this change detection algorithm. This thesis researches if a localization of the change is possible in regards of needed effort and quality.

### 1.3 SCOPE

In this master thesis only change detection for polygons with the ML-technique of XGBoost is applied. The polygons and their shapes are already predefined by the digital map and will not be changed or redrawn. The complete method is seen as a support for the manual change detection and is not intended to replace it. Furthermore, this thesis will not compare different ML techniques.

### 1.4 THESIS OUTLINE

This thesis is structured in the following way:

**Chapter 2** introduces the theoretical background that is necessary to understand the workflow and methods of the thesis. This includes the creation of the input data, the basic concept of ML and especially the concepts of Gradient boosting and XGBoost. Furthermore, the theoretical background of the features is explained.

**Chapter 3** gives an overview about the current state of the research in the topic of change detection in a geographical context.

**Chapter 4** describes the methodology of the change detection for a digital map using aerial images and point clouds.

**Chapter 5** describes the actual implementation of the algorithm with the exact settings. Design decisions are explained in this chapter.

**Chapter 6** describes the results of the algorithm and comparison to existing methods. A small chapter with recommendations for reproducers can be found here.

**Chapter 7** presents the conclusions drawn from this thesis and answers explicitly the research questions. Furthermore, the most important contributions of this thesis and future work are discussed.



# 2 | THEORETICAL BACKGROUND

This chapter gives a short technical background for the data and the concepts used in this thesis.

§ 2.1 explains the basics of the input data and how they are created. The most important information source for the BGT is the official website of the [Ministerie van Binnenlandse Zaken en Koninkrijksrelaties \[2018\]](#) (Dutch Ministry of Interiors). The other main sources for the input data are [Ruzgiene \[2012\]](#) for aerial images, [Vosselman and Maas \[2010\]](#) (LIDAR) and [Szeliski \[2011\]](#) (Photogrammetry) for point clouds. Even though the input data was provided from the company, it is crucial to understand the basics of their creation, as that can give important information for the later feature selection for the ML-algorithm.

§ 2.2 gives a small introduction to ML. The basic concepts are explained mainly with web resources. There are many different blogs and high-quality developer guides available, for example the online developer guide for ML from Amazon<sup>1</sup> or Kaggle<sup>2</sup>. For the explanation of gradient boosting (§ 2.3) and XGBoost (§ 2.3.3) the YouTube-videos of Statquest of Josh Starmer<sup>3</sup> provide excellent information. Even though the algorithms are already implemented in the XGBoost library, understanding the concepts of XGBoost is necessary to understand the results and in order to improve them.

In § 2.4 small explanations of the technical background of the features are given. Different sources were used and are described in the respective chapters.

## 2.1 INPUT DATA

The input data for this thesis consists of three different data sources:

1. Digital map consisting of polygons (§ 2.1.1)
2. Aerial imagery (§ 2.1.2)
3. Point cloud (§ 2.1.3)

Representative for the digital map, the creation and attributes of the BGT are described. For the algorithm used in this thesis the point cloud will be converted into a Digital Surface Model (DSM). Therefore, a small explanation of this model and how it is created will be given in § 2.1.4.

### 2.1.1 BGT

The BGT is a digital map describing the surface structure of the entire Netherlands. As can be seen in figure 2.1 it consists of categorized polygons each having a unique id and describing the size and shape of a real life object. Furthermore, each polygon is geo-referenced with the accuracy of these polygons up to 20 cm as described by [Ministerie van Binnenlandse Zaken en Koninkrijksrelaties \[2018\]](#). There are many different classes for the polygons to describe the surface as accurately as possible.

---

<sup>1</sup> available at [Amazon](#).

<sup>2</sup> available at [Kaggle](#).

<sup>3</sup> available at [Statquest](#).

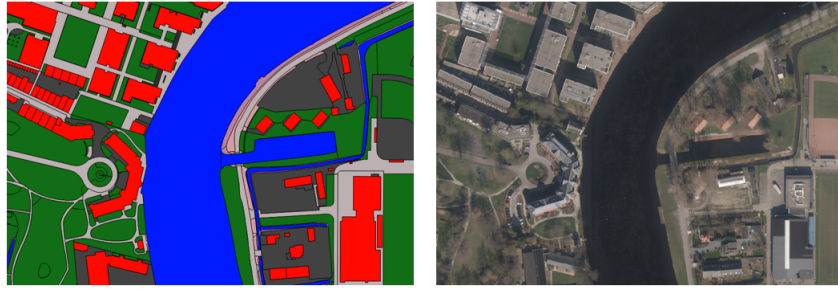


Figure 2.1: BGT and aerial image of the same scene

According to the official website [[Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, 2018](#)] following main categories of classes are recorded:

- buildings: all kind of buildings (residential as well as industrial)
- roads: all kind of streets, bike-lanes and sidewalks
- water: lakes, rivers but also canals
- green: all kind of vegetation (forests, agriculture, parks, etc..)
- railway lines: the entire rail network of ProRail
- works of art: bridges, tunnels and other objects

The basic infrastructure for the [BGT](#) (publishing and communication) is handled by one authority (Publieke Dienstverlening Op de Kaart, PDOK). However, there are many more stakeholders responsible for creating and updating the [BGT](#). Each municipality, the Rijkswaterstaat (Ministry for Infrastructure), the Dutch Railway company and many more are involved in this process with different responsibilities. They have to supply the data for this digital map according to standardized agreements for classes and quality. Even though agreements exist, due to this high number of data providers the accuracy and exact handling of different classes (which class should be assigned to a certain object?) is still different throughout the Netherlands.

Many people or companies rely on the [BGT](#) and it is often used as the basis for further applications. These include for example architects using the digital map as an additional instrument for designing buildings <sup>4</sup> or private companies using it as input data <sup>5</sup>. For authorities in the Netherlands the [BGT](#) is even more important. They are obliged by law to use this digital map in all projects that require a map as a foundation [[Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, 2017](#)]. Example projects are listed on the official website and include urban planning, visualizing statistics and many more processes.

### 2.1.2 Aerial images

As the name already specifies, aerial images are images taken by airborne vehicles like airplanes, helicopters or, as the most recent developments in this field, drones. The most common vehicles for aerial images of complete cities are still airplanes, as they can usually cover the whole area in a single flight and can carry the weight of the cameras needed for a sufficient resolution. Not every type of aerial image can be used for this thesis. Multiple requirements must be fulfilled with some general requirements following [[Ruzgiene, 2012](#)]:

<sup>4</sup> see [here](#).

<sup>5</sup> see [here](#).

- geo-referenced: The coordinate system as well as the exact position of every pixel of the image must be known.
- visible: No clouds or other obstacles should be in the image, a clear view on all objects is required.
- overlapped: In order to merge the multiple images, they must overlap.

Furthermore, these images can be specified in two different sub-categories (compare [Hu et al. \[2016\]](#)):

- Ortho-image: A geo-referenced photograph of the surface with a uniform scale.
- Trueortho-image: An ortho-image with all vertical features re-projected and no visible lean.

The difference between both image types can be seen in figure 2.2. In the ortho-images the houses are not completely oblique and small parts of the front facade can be seen (in the red circle). The Trueortho-image instead is corrected and no facade can be seen. These images are created by merging multiple overlapping ortho-images. From each image only the non-oblique parts are taken.



Figure 2.2: Ortho-Image (left) and trueortho-image (right) of same scene at different time steps

Another important feature of the aerial image is its resolution. In order to get meaningful results and assure a change detection even for smaller objects it is important to have a good resolution of the aerial images. Resolution in this case is defined as the distance between two pixels. The higher the resolution the more pixels can be found for the same area. To achieve a change detection for an object, it must be possible to extract enough information for this particular object. As [Cai \[2003\]](#) describes in his paper, at least around 1000 pixels are needed for humans to detect objects with tendencies to higher numbers of 2000 pixels for more complex objects. These numbers should be reached for all considered polygons in this thesis as well.

### 2.1.3 Point cloud

A point cloud is a set of points in the space each having coordinates usually setting the position in  $x$ ,  $y$  and  $z$ -direction. Multiple points together form a set of points that can describe objects or natural landscapes in 3D. In this case, the point cloud describes the surface structure of the research area with  $x$  and  $y$  as the exact position of the point and  $z$  as its height.

There are two main techniques to create a point cloud of the surface: LIDAR and photogrammetry.

- LIDAR

LIDAR (Light Detection And Ranging) describes a method in which laser beams are used to extract information about the profile of a surface. It exploits the fact that the speed of light is a natural constant with a velocity of 299.792.458 m/s. The LIDAR scanner emits a laser beam to a point on the surface. There the beam gets reflected and returns to the emitter. Both times, when sending the laser and when it is returning, are saved. With this information the exact distance between the scanner and the surface can be calculated.

The most common way to get large scale point clouds for urban areas is airborne laser-scanning. Similar to the creation of aerial images, the scanning device is mounted to an airborne vehicle (mostly airplanes) together with a high precision GPS and timing device. The process of scanning is done repeatedly with the rays being emitted in different directions.

The advantages of LIDAR data are its high accuracy and high density of points. Only few processing is needed after collecting the data and the point cloud can be used almost instantaneously. Next to the height values, additional information is available due to some physical attributes of LIDAR (like first and last return or point density, for more information about this topic see [Scaioni et al. \[2018\]](#)).

- 3D Photogrammetry

A newer approach for creating point clouds is 3D photogrammetry, in which the 3D information of an object or surface is extracted from 2D images. It can be used to create point clouds for complete cities and landscapes using overlapping aerial images. Image pairs, two images capturing the same scene from different angles, are used to gather 3D information in a process called computer stereo vision. Readers with interest in this topic are referenced to [Szeliski \[2011\]](#).

In a simple view, three main steps must be taken to create a point cloud from overlapping aerial images:

1. Rectification

To simplify the complete process the images must be rectified so that the image plane is equal. Depending on the source of the images, this step is not necessary. For many aerial images the camera is mounted in a way that the photographs have the same image plane already.

2. Stereo matching

Correspondent pixels in the image pairs must be found to determine in which relation the images are located to each other. As the images are rectified the corresponding pixels are located on the same line. Many different algorithms exist for this tasks, with the simplest one matching the images by pixel patches up into more advanced algorithms using neural networks for stereo matching. The distances between matching pixels are saved and can be also visualized in the form of a disparity map.

3. Triangulation

In this step the transformation from 2D to 3D information is happening. Not only the disparity values but also exact information from the camera and the pictures are needed. This includes focal length of the camera and the exact positions of the camera when making the image pairs. With all this information the z-position (height) of a point in the image can be calculated using matrix equations.

When used for airborne imagery, this process is also known as stereo photogrammetry. Airborne images are taken in parallel movement and in so-called blocks, roughly rectangular areas. Usually the image overlaps 30% to the side and 60% to the forward [Kraus, 1997].

#### 2.1.4 DSM

Even though it is possible to use the height information from the points of the point cloud directly, it is a common approach to convert the point cloud into a DSM. It is an elevation model that includes the surface of not only the ground but everything located on it, for example buildings, trees or cars [Zhou, 2017].

Like for an aerial image, the DSM is a raster in which each cell contains the height instead of colour information. The advantage of having this model as opposed to a point cloud is that information can be extracted in the same way as for the aerial images.

## 2.2 MACHINE LEARNING

The field of machine learning is a preamble for many different techniques where an artificial system learns with data and tries to derive general rules from examples. The focus and central point of ML is the process of learning. Already Mitchell [1997] describes learning as follows:

*A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .*

In the context of this thesis, task  $T$  is the change detection with performance measure  $P$ , which could be for example the relation between correctly and incorrectly recognized changes. Experience  $E$  are the provided polygons with labeled changes (specified if a polygon is changing or not). The tasks  $T$  can be summarized into a model, which describes the way of getting from the input data to the desired output (probability of change). Training is the process of adapting this model so that it can predict the output correctly.

Some key words are important to know when describing the input data for ML [Bhattacharjee, 2017]. An entry describes one entity of the dataset, in the case of the thesis, one single polygon. A feature is defined as an individual measurable property of an entry. The number of features used is the dimension of the data. Note that the features of a polygon are derived separately for both years. The target values are the outcome for every entry that the model should predict, in this case it would be the probability for every polygon that it is changing between the time-steps.

While training a model, there is always the risk of under- and overfitting the model [Amazon, 2020]. Figure 2.3 displays both scenarios.

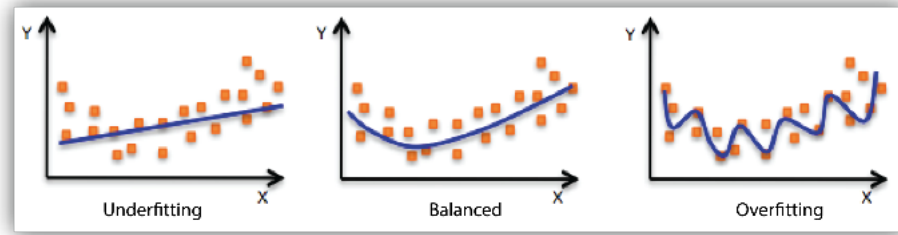


Figure 2.3: Comparison of underfitting, balanced and overfitting (© Amazon Machine learning)

The ultimate goal of a ML algorithm is to find the balanced model. The general trend of the training data is recognized and new data can be predicted correctly. The underfitting model performs poorly on the training data and is unable to capture the relationship between  $X$  (input data) and  $Y$  (target values). That can happen when the model training was stopped too early or the model is too simple (too few features). The opposite is overfitting. The model fits perfectly to the example data and can predict the target values precisely. However, this model is trained to predict exactly the values from the example and did not capture the general relationship between  $X$  and  $Y$ . It can be caused by having models, that are too specific for the existing data.

### 2.2.1 Learning methods

Three main types of algorithms can be distinguished in ML as described by Dwivedi [2019]: Supervised vs unsupervised learning vs reinforcement learning. The first step when applying a ML technique is to identify the learning type needed.

- Supervised learning describes a setting in which labeled data is already available. The target values that should be predicted are already known and the challenge is to label correctly: the input data with unknown ground truth must be sorted into the different labels.
- For unsupervised learning labeled data is not available. Only the raw data is used to detect patterns or cluster the data. There are no right or wrong answers, only more or less precise patterns or clusters.
- In reinforcement learning no raw data is given to the algorithm and instead an agent interacts with its environment. It must explore the environment and find which tasks give which rewards. A lot of trial and error is involved in this process. It is mainly used for robotics, gaming and navigation. This task is different from the other two tasks as it cannot be used for the prediction of target values.

### 2.2.2 Categories

An important step when applying ML for a problem is to define the required task. Depending on the type of ML algorithm and the target data, a different task is required [Soni, 2019], as can be seen in table 2.1:

|            | Supervised     | Unsupervised             |
|------------|----------------|--------------------------|
| Discrete   | Classification | Clustering               |
| Continuous | Regression     | Dimensionality reduction |

Table 2.1: Overview machine learning tasks

Discrete data can only take certain values (for example categories), whereas continuous data can take any value.

In classification and regression, specific relationships between features and target values must be found to label the input data. In classification the target values have determined categories that can be used for labeling (for example change or no change), whereas in regression data will be labeled with continuous values (for example the height of a building).

In clustering, the goal is to learn the inherent structure of the data without having to provide explicit labels beforehand. In dimensionality reduction it is the goal to discover the relationships between individual features. This allows to represent the data using only the interrelate features and therefore reduce the number of columns without losing information (a dataset that of 100 features can be reduced to one with 50 columns).

## 2.3 GRADIENT BOOSTING

Gradient boosting is a [ML](#) technique that can be used for both classification and regression. XGBoost follows the principles of gradient boosting with some differences in modeling details. Therefore, for the understanding of XGBoost it is important to understand gradient boosting beforehand.

Boosting can be described as a process in which weak learners are combined [[Mayr et al., 2014](#)] into a strong learner that can reduce both bias and variance (in the context of [ML](#); a description can be found in § 4.4). A simple example for boosting would be a scenario, in which a set of candidates should be classified if they like computer games or not. One weak learner, like gender of the candidates (with the assumption that males like computer games and females not), would already give a first estimation but would lead to many incorrectly classified candidates. Including more weak learners (age, study direction, internet usage times) can improve the results.

For both the description of decision trees and the description of the functionality of gradient boosting, the YouTube-tutorials of Statquest were used. For more extensive information to gradient boosting the reader is referred to the paper of [Natekin and Knoll \[2013\]](#).

### 2.3.1 Decision Trees

For the following description of Gradient boosting and XGBoost it is eminent to understand the concept of a decision tree and how it is constructed. A decision tree is a directed graph that depicts every possible outcome of a question/decision. It always has one root node, usually multiple internal nodes and multiple end leaf nodes. In [figure 2.4](#) a simple decision tree is displayed that detects if a polygon has changed. Every leaf can have a different number of decision levels and nodes can be used multiple times. The blue boxes are the separators, the black values describe the condition and the red/green boxes contain the final outcome.

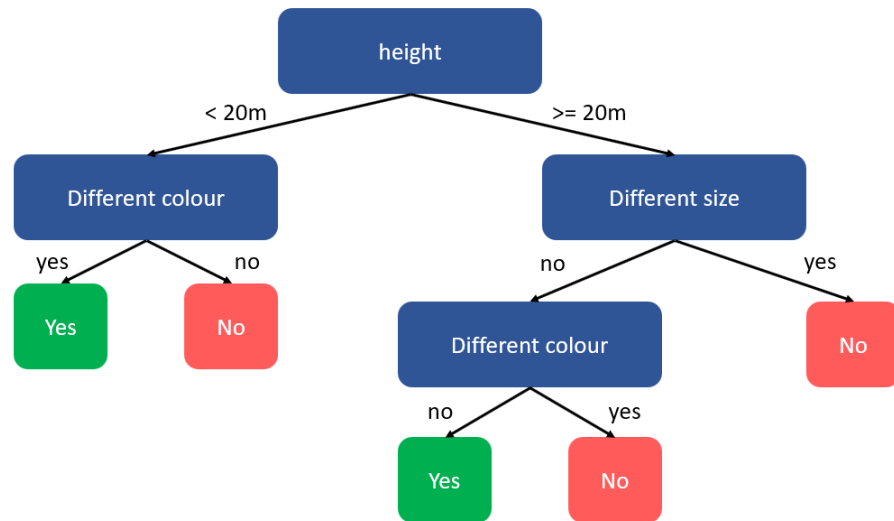


Figure 2.4: Example for a decision tree

Many ML algorithms use decision trees to predict the outcome values. When designing complicated decision trees, including multiple features and leaves, the order of the features plays an important role, for example which feature should be the root node. A common way to determine the order is calculating the impurity for each feature. Impurity is a value that describes how many different target values are in an end leaf. An end leaf that has only target values of one class is considered as pure. There are multiple ways to calculate this value, a common approach is using the Gini-impurity. Following steps describe the procedure of creating the decision tree:

1. A simple decision tree for each feature is created. For a Boolean field it is a simple yes/no question, for a categorical field each category is tried out. Numerical values are ordered ascending and the average weight between each entry is taken and a tree is created.
2. For each leaf node of the tree, the number of true and false values are counted. If both true and false values are in the same leaf (which is usually the case), they are considered as impure. The exact amount of impurity can be measured with formula 2.1:

$$impurity = 1 - \left(\frac{n_{True}}{n_{True} + n_{False}}\right)^2 - \left(\frac{n_{False}}{n_{True} + n_{False}}\right)^2 \quad (2.1)$$

3. To get the total value of impurity for one tree, the values of both leaves must be combined using the average. As both leaves can represent a different number of entries, the weighted average must be calculated.
4. The decision tree with the lowest impurity score divides the dataset the best. The corresponding features should therefore be considered earlier and can be found at the top of the decision tree.
5. To get more levels for the final decision tree the process is repeated with the entries left in the end leaf node of the decision tree with the lowest impurity score.



### 2.3.2 Functionality

Gradient boosting incorporates many weak learners and decision trees to fit a model based on training data. Creating the model consists of multiple steps used both for regression and classification with only some minor differences. The creation of the model will be explained with a small exemplary dataset, consisting of three features that should predict the outcome feature, e.g. classify if the entry is a building or not. Table 2.2 displays the example data:

| id | squared shape | colour | height (m) | building |
|----|---------------|--------|------------|----------|
| 1  | yes           | red    | 12         | yes      |
| 2  | yes           | gray   | 87         | yes      |
| 3  | no            | red    | 44         | no       |
| 4  | yes           | green  | 19         | no       |
| 5  | no            | gray   | 32         | yes      |
| 6  | no            | red    | 14         | yes      |

Table 2.2: Example data for classification using gradient boosting

1. As a first step an initial prediction value is calculated. If the outcome feature is continuous, just the average value of the outcome feature is taken. For a categorical outcome feature, the initial prediction value is calculated with the likelihood using formula 2.2:

$$prediction\ value = \frac{e^{\log(\frac{n_{True}}{n_{False}})}}{1 + e^{\log(\frac{n_{True}}{n_{False}})}} \quad (2.2)$$

In this case the probability of an entry being a building is 0.7.

Then a so called pseudo residual is calculated for every entry by subtracting the initial prediction value from the real outcome value as shown in table 2.3. If the outcome value is not continuous, 1 or 0 are used to replace the true and false values. For the exemplary dataset the following values are calculated:

| id | outcome | initial prediction value | pseudo residual |
|----|---------|--------------------------|-----------------|
| 1  | yes (1) | 0.7                      | 0.3             |
| 2  | yes (1) | 0.7                      | 0.3             |
| 3  | no (0)  | 0.7                      | -0.7            |
| 4  | no (0)  | 0.7                      | -0.7            |
| 5  | yes (0) | 0.7                      | 0.3             |
| 6  | yes (0) | 0.7                      | 0.3             |

Table 2.3: Exemplary initial prediction value and pseudo residual

- Next, a decision tree with the three available features is created to predict the pseudo residuals. In the end leaves, the pseudo residuals of each entry, that follow the path to this leaf, are saved. The rules for creating the tree that describes the data best, are described in § 2.3.1. Usually the number of leaves for that tree are limited, with the number of leaves influencing the end results. In this example, the number of leaves is limited to three.

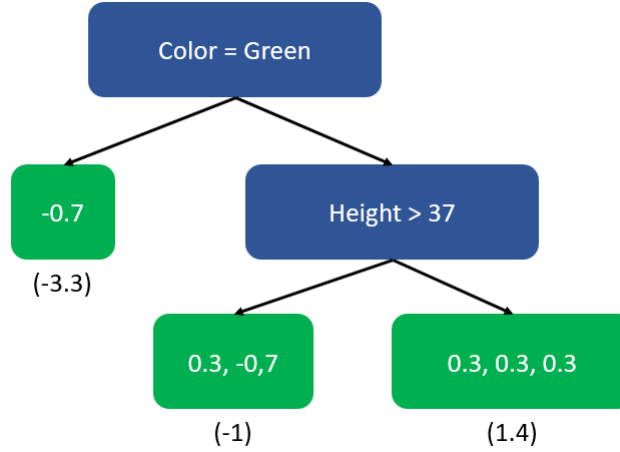


Figure 2.5: decision tree for predicting the residuals (output values in brackets)

Figure 2.5 displays the decision tree for predicting the residuals. As the number of leaves is limited, values from buildings and no buildings can be seen in the end leaves. The model is not perfect yet, as there are still different target values in the end leaf.

- For each end leaf an output value must be calculated. If the outcome features are continuous, the output value is the average value of all values inside an end leaf. For categorical outcome features, the output value can be calculated with a transformation using formula 2.3:

$$output\ value = \frac{\sum residual_i}{\sum [prev.\ prob._i * (1 - prev.\ prob._i)]} \quad (2.3)$$

When the first tree is built, the previous probability refers to the probability of an initial leaf. Figure 2.5 displays the output values for the example in brackets.

- Following, the predictions of table 2.3 are updated by combining the initial probability with the leaves of the decision tree. When adding the values of the decision tree usually a learning rate is applied to scale the contribution of a single tree and minimize the influence. Thus, if a decision tree does not split the entries well it will not cause a lot of false classified data. Formula 2.4 depicts this process. For this example, the learning rate is set to 0.8. Table 2.4 displays all values for each entry.

$$new\ pred.\ Value = old\ pred.\ value + (learning\ rate * output\ value) \quad (2.4)$$

Similar to step 1 the new prediction must be converted into a probability using formula 2.5:

$$probability = \frac{e^{prediction}}{1 + e^{prediction}} \quad (2.5)$$

| id | initial pred. value | output value | new pred. value | new Probability |
|----|---------------------|--------------|-----------------|-----------------|
| 1  | 0.7                 | 1.4          | 1.8             | 0.9             |
| 2  | 0.7                 | -1           | -0.1            | 0.5             |
| 3  | 0.7                 | -1           | -0.1            | 0.5             |
| 4  | 0.7                 | -3.3         | -1.94           | 0.1             |
| 5  | 0.7                 | 1.4          | 1.8             | 0.9             |
| 6  | 0.7                 | 1.4          | 1.8             | 0.9             |

Table 2.4: Probability values for dataset

- With many entries having now different probabilities, new residuals can be calculated. These residuals can be used to build a new tree. The new prediction values are applied again with the learning rate to the old values. This continues until the maximum permitted number of trees or the residuals are smaller than a threshold value.

As can be seen in Figure 2.6, all trees together form the model that is applied on new data to calculate the target values.

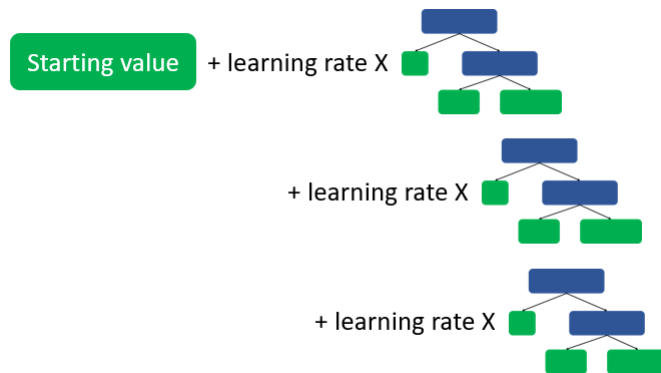


Figure 2.6: Creation of new model with decision trees and learning rate

### 2.3.3 XGBoost

XGBoost started as a research project at the University of Washington [Chen and Guestrin, 2016] and is short for eXtreme Gradient Boost. It is an implementation of gradient boosting machines and it is currently the last step in the evolution of tree-based decision systems as shown in figure 2.7. It is an open source software library that is used for many ML-tasks.

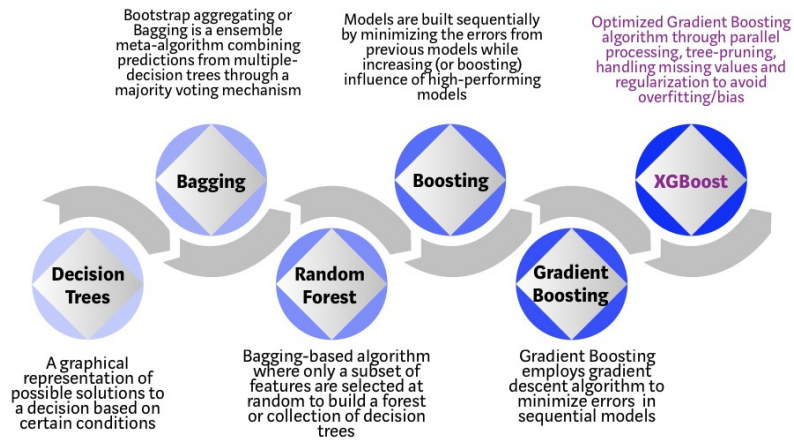


Figure 2.7: Evolution of tree-based decision systems (© Vishal Morde)

It is used in this thesis for implementing the model for change detection and selected for the following reasons:

- According to several benchmarks (for example of Pafka [2020]) XGBoost is one of the fastest ML-algorithms. This includes both the time for training the model as well as applying the model to unknown data. It scales well with the number of objects to classify and also the number of features has a small influence on computation times.
- It can be used within the python environment and is well documented.
- The importance of features can easily be derived from the model and there are functions within the package to display this importance. This helps to select only meaningful features for the change detection and get rid of features that only have a very low impact.
- XGBoost has many built-in mechanics (for example pruning the decision trees) that makes it very resilient to overfitting.

Even though it shares many similarities with gradient boosting, it is different in the details. The basic concept of building trees and adding them to calculate the target values is the same. The biggest difference is how the trees are built. Like Gradient boosting in the previous chapter, XGBoost will be explained with example data and with a focus on the differences to Gradient boosting. The maximum tree level for this example is two. Table 2.5 displays example data:

| id | height difference (m) | change |
|----|-----------------------|--------|
| 1  | 4                     | no     |
| 2  | 8                     | yes    |
| 3  | 12                    | yes    |
| 4  | 18                    | no     |

Table 2.5: Example data for XGBoost

The following steps are taken to create a model in XGBoost:

1. Again, an initial prediction value and the residual of each value must be calculated. For XGBoost for both regression and classification it is always set to the same value, 0.5 being the default. True and false values are again replaced by 1 and 0 and the residuals are calculated.

2. Like at gradient boosting a tree must be designed. In this case however all residuals are put together in one starting leaf to calculate the so-called similarity score formula 2.6. Lambda is a regularization parameter and is 0 in this example.

$$\text{Similarity score} = \frac{(\sum \text{Residuals})^2}{\sum(\text{prev. prob.} * (1 - \text{prev. prob.})) + \lambda} \quad (2.6)$$

For the example data the similarity score would be 0, as the two positive and the two negative values cancel each other out. Afterwards it must be checked if there is a better prediction. Usually the average value of two following entries is taken.

3. Figure 2.8 displays an example-tree with a split based on the average value between feature 3 and 4. In the brackets the similarity scores are displayed.

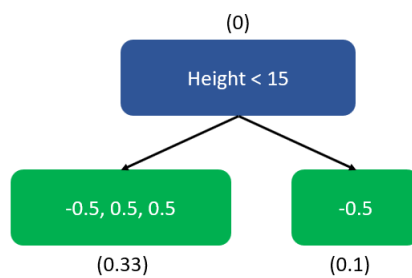


Figure 2.8: Decision tree (similarity scores in brackets)

To quantify how much better the new leaves cluster similar residuals, the gain must be calculated using formula 2.7.

$$\text{gain} = \text{Similarity}_{\text{Left}} + \text{Similarity}_{\text{Right}} - \text{Similarity}_{\text{Root}} \quad (2.7)$$

The gain for this tree would be 1.33. No other threshold is giving a higher gain, thus this split is the right one.

4. As the left end node has multiple residuals it is possible to split it again like before by selecting the split with the maximum gain. In this case the split would be performed with height < 6 as here a maximum gain of 2.66 can be found. In figure 2.9 the new tree can be found. As the tree-level is limited to 2, no further leaves are added.

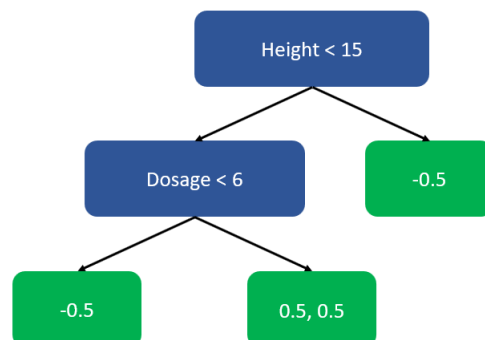


Figure 2.9: Decision Tree with end leaves

5. In contrast to gradient boosting, XGBoost also includes some limitations regarding the trees, including cover, pruning and regularization. All processes simplify already available trees to make the algorithm more resilient to overfitting. With cover a minimum number of residuals per leaf can be assured. Pruning means that every new leaf and the gain is checked if their result exceeds a certain limit (a parameter called gamma). If this is not the case, the leaf is pruned. When raising lambda (regularization parameter), the gain of the leaves will decrease and pruning is more likely.
6. Like in Gradient boosting new residuals are calculated using the already known formulas described in § 2.3.2.

This process of building trees from value, simplify them and calculating new residuals is repeated many times until the maximum number of allowed trees. By default, these trees are built with trying out all features and select every time the features that would give the maximum gain. Like in gradient boosting, a model is built from the trees. s Contrary to the standard gradient boosting method some more Quality of life (QoL)-features are implemented with some important ones mentioned the following, such as [Brownlee, 2016a]:

- Parallel processing is possible
- built-in handling of missing values
- built-in randomization

## 2.4 FEATURES

The term "feature" describes an attribute of a single polygon and its correspondent geographical extent on the aerial image or the point cloud in this thesis. Many features are collected as weak learners and are the foundation for the ML change detection. This section describes the features used in this thesis.

### 2.4.1 Colour features

Colour features describe the aerial images itself. They are easy to derive but very susceptible to small changes (A street with cars has different values as a street without cars).

#### RGB

The provided aerial images - like most digital photographs - consist of three different bands: red, green and blue. Each band describes the intensity of this particular colour for each pixel of the image. Depending on the intensity of each of red, green and blue a different mixed colour is present. The values are easy to derive from images and many algorithms exists for this colour scheme. However, it is strongly influenced by shadow [Jang et al., 2019].

#### HSV

Like Red Green Blue (RGB) the Hue Saturation Value (HSV) colour space consist of three different bands: Hue, saturation and value. Instead of defining the colour with the intensity of a certain colour, it is described via Hue (which colour), Saturation (how intense is the colour) and value (how light/dark is the colour). It is more similar to how humans perceive colour. The advantage of HSV is that shadow only influences the value-band and not the other two bands.

### 2.4.2 Height features

Height features are derived from the point cloud or the DSM. Furthermore, two other features can be directly derived from the height:

- Slope describes how fast the height is rising between two points.
- Aspect is related to slope but describes the direction of the slope.

### 2.4.3 Polygon features

Instead of looking at the values from the aerial image or the point cloud, attributes from the polygon itself are used to extract features. Simple attributes like the size or shape of a polygon, but also derived features like the compactness (how similar is the shape of polygon to a square) of the polygon can give valuable information. Furthermore the original category of the polygon can be used as an input.

### 2.4.4 Progressive features

This term describes features in which the feature value is not directly derived from the input source, but must be processed first. Some features require gray-scale images, so that the aerial images must be converted beforehand.

#### Shadow calculation

Humans can recognize shadow when looking at an object, for the computer this is more difficult. The colour information of a polygon with shadow is different and that must be considered for the change detection.

For all aerial images used in this thesis the exact position as well as date and time, where and when the image was taken is known. This information is used to recalculate the exact position of the sun (based on [Bhattacharya et al. \[2019\]](#)). Furthermore, a height model of the area is also available. Both pieces of information can be combined to recalculate exactly locations that can be expected being in the shadow. This information can be used as a feature for the ML algorithm by calculating the percentage of shadow pixels in a polygon.

However, it is important to note that this method cannot deliver perfect results: Shadows created by buildings or other opaque structures are reconstructed in a better way than shadows from trees. Trees allow some light to shine through, thus their created shadow is not completely shaded but more dim.

#### Haralick features

These features were originally developed for analysis of medical images but are now commonly used for all kind of image analysis by assessing the texture in images. Gray level co-occurrence matrices are used as a method to quantify the spatial relation of neighbouring pixels in an image.

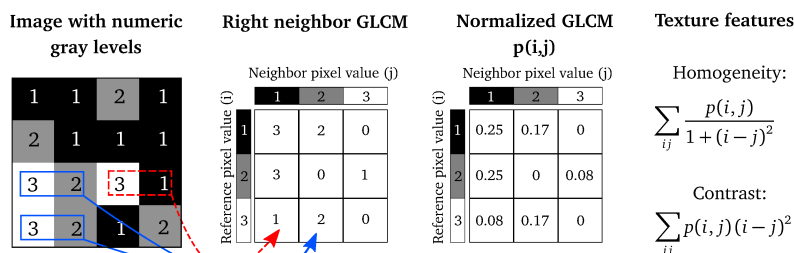


Figure 2.10: Functionality of Haralick features (© Löfstedt et Al.)

Haralick features are calculated by counting the spatial relation between the intensity of gray values in the image [Brynnolfsson et al., 2017]. Figure 2.10 depicts a simple example:

1. A matrix with the size  $N \times N$  must be created, in which  $N$  is the number of different gray level values in the image, in this case  $N=3$ .
2. For every combination of two colors the combination of gray level pairs in the image is counted for a certain direction. In this case the direction is from left to right, with the reference pixel always being the column and the neighbouring value being the row. In this case, only once a gray value of 1 directly right at a gray value of 3 (red) is found and this value is therefore 1. A pixel value of 2 however is located two times directly right of a gray value of 3, so that this value is 2 (blue). These calculations are done for every combination. In this image only the direction from left to right is considered, however in total 8 directions and 8 different matrices are possible.
3. For further usage the matrix must be normalized. Standard practice is to calculate an average from all matrices to get an rotational invariance.
4. This matrix can be used to calculate multiple statistics. In figure 2.10 two example features (Homogeneity and Contrast) are displayed <sup>6</sup>.

### Fourier transform

With this method images are decomposed into its sine and cosine components. For digital images the Discrete Fourier transform is used. Every image is translated in a Fourier transform that is unique for an image. However, similar images have similar Fourier transforms. These images can then be presented through a corrugation function a 3D-curve in which peaks and valleys are found (see figure 2.11). The number of peaks and their structure can be measured and used as features. For more information please refer to Popa and Cernăzanu-Glăvan [2018].

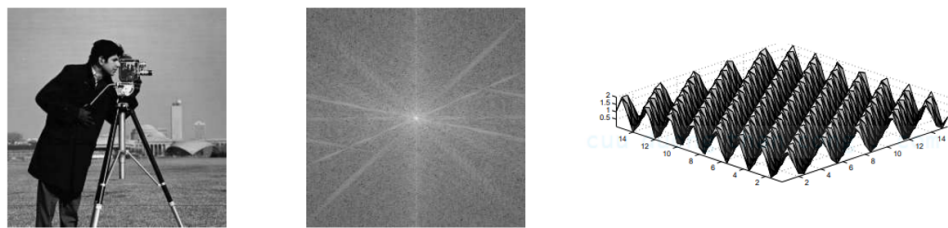


Figure 2.11: Image and its Fourier transform and 3D curve

### Local binary pattern

Local Binary Patterns (LBP) are visual descriptors used for image classification in computer vision. They are used to distinguish between different surface textures and convert them into numeric representations. Whereas Haralick features are consisting of a global texture representation, LBP are considering local representations of the textures [Rosebrock, 2020].

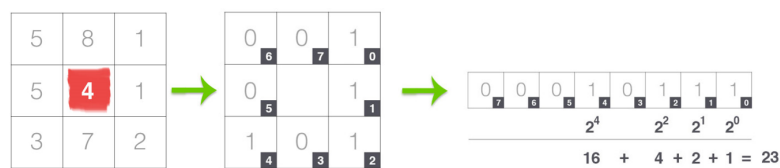


Figure 2.12: Functionality of Local binary pattern (© Rosebrock)

<sup>6</sup> A complete list of features can be found [here](#).



Figure 2.12 shows the basic principle of this method: A small window is put at every pixel of the gray-scale image. If the pixel at a certain position has a higher intensity, the corresponding value is set to 0, if the intensity is lower, the corresponding value is set to 1. This window can be stored as a vector of the length 8 and therefore be converted to a decimal (transformation from binary representation). The range of numbers that can be represented with 8 digits is from 0-255. The values of all numbers of all pixels are displayed as a histogram. The values of this histogram (a 256-sized vector) are used as features for XGBoost.

Adaptations of LBP can be made with different numbers of pixels that are counted and different sizes of the window. Furthermore can the size of the vector be reduced by a technique referred to uniform patterns [Ojala et al., 2002].

# 3 | RELATED WORK

The development of change detection as the process of identifying changes between objects or scenes in different time series coincides with the history of remote sensing [Théau, 2008]. It is used in many applications for research, business or management. Representative three papers from different fields using change detection are displayed to present that change detection is used in many different disciplines of both physical and human geography and with different approaches:

- crop monitoring: de Bie et al. [2008] incorporate change detection techniques to identify the extent and nature of land cover units. On satellite images of the same region in a 10-day period they compared unsupervised classifications. Different profiles of changes are used then for identification.
- urban growth: Hegazy and Kaloop [2015] use change detection to map the urban growth and the change in the land usage. Remote sensing imagery is used for classification of the surface and simple change detection methods are applied. This information of how the surface is changing can be used for optimized urban planning.
- large scale urbanization: Taubenböck et al. [2012] are using object-oriented classification on remote-sensing imagery to do a worldwide monitoring of urban area. Afterwards pixel-based change detection is used to get insights for urbanization, risk management or population assessment.

With the higher availability of remote sensing data and increasing performance of computer systems more algorithms for change detection were developed. Already around 1990 Singh [1989] presented an overview of different methods for change detection using remote sensing images, including pixel-based comparisons or the usage of indices like NDVI. Since then, change detection is a recurring topic of research with many different methods. A comprehensive overview about change detection techniques is given by Lu et al. [2004]. Ban and Yousif [2016] created an updated overview for change detection algorithms with the focus on both optical and SAR-images. The recent development of incorporating 3D information for change detection is considered in the overview of Qin et al. [2016].

The two major data sources used for change detection are satellite images and aerial imagery from UAV/airplanes. Even though satellite images can include valuable information in their additional bands and more algorithms exist for these kind of images, they have some disadvantages especially for urban change detection: The access to these images is more limited and expensive and the resolution especially for urban scenes is usually too low. For these reasons mainly research based on aerial imagery will be discussed in this chapter.

For urban scenes many researchers focus their change detection on certain types of objects, in most cases buildings [Nebiker et al., 2014; Pang et al., 2018]. Due to their structure and attributes (mostly squared with a certain height) identifying and applying change detection for buildings is easier than for example streets and buildings are usually of higher economic interest. Less papers can be found for road detection, as this object type is more difficult to identify and to apply change detection to. However, there are papers handling change detection for roads, such as Song et al. [2005].

To the contrary of the approach presented in this thesis, many studies apply change detection directly to the aerial images and do not have a digital map as an indicator of the original surface structure. Instead the objects for change detection must be segmented beforehand. However, few papers have a similar approach with digital maps as an additional data source [Knudsen and Olsen, 2003] or are even only using digital maps [Zhou et al., 2018]. The methodology of merging their input sources with the digital map can be transferred on this thesis as well.

In general, three main categories of change detection methods can be distinguished and will be discussed in separate chapters of this thesis:

- 3.1 Traditional methods using only images as input
- 3.2 Height methods using images and height information as input
- 3.3 Machine-learning based methods

Finally, chapter 3.4 will give a conclusion of the relevant work with a focus on how the methods can be incorporated for this thesis.

### 3.1 TRADITIONAL METHODS

These methods were the first to be developed and only use aerial images as an input for change detection. Many algorithms originally developed for satellite images can be used for aerial images as well. The whole topic of change detection with images as an input has been explored by many researchers and there are many different approaches available. As Hussain et al. [2013] describe in their paper, the trend in change detection is from simple pixel-based methods to more object-based approaches [Peng and Zhang, 2017; Shi et al., 2012]. Many of these papers describe algorithms to extract information from the image that also can be used as features for ML algorithms.

A good introduction and overview about pixel-based methods can be found in the book of İlsever and Ünsalan [2012]. They are describing multiple methods for change detection based on the pixel values itself. Furthermore, they introduce texture-based descriptors by using a grey-occurrence matrix. Several of the methods described can be used directly in this thesis to create features as input for the ML-algorithm. However, pixel-based comparison are becoming less common as change detection is moving towards object-based comparisons due to better results. These methods are a good source for features that can be used for our algorithms:

- Benedek and Sziranyi [2009] propose a probabilistic model to detect changes in aerial images from different years and seasonal conditions. Information from three different observations (statistics of global intensity, local correlation and contrast) is utilized in a conditional Mixed Markov Model. Changes that are statistically unusual are detected and considered as real changes. They achieve an overall accuracy of 90 to 95%, however their approach is more suited to change detection for larger regions and less for urban scenes with smaller objects. Especially their ideas for correlation and contrast could be used as features in this project.
- Rowe and Grewe [2001] are describing a system of automatic change detection especially for buildings and roads. They are extracting line segments with the canny-edge-detector for different input images and are cancelling out overlapping edges. The left-over edges are a good estimation for changes and can be used as a support for manual change detection. Their idea of using canny-edge-detection for identifying changes could also be used as an additional feature for the ML algorithm.

- Pang et al. [2018] propose a method for object-based change detection in which the objects are created by watershed segmentation and multiple features are extracted based on these objects with a special focus set on local binary patterns. These patterns could be used in this thesis as well.

## 3.2 HEIGHT METHODS

As an additional input these methods also use height information next to the aerial images for change detection. Height is a very good indicator to detect changes, especially for buildings and is usually derived from point clouds. These methods first came up when LIDAR was available, as before acquiring point clouds in bigger scale was not possible.

In the last few years 3D photogrammetry became more common as an additional source for point clouds. Even though it lacks some additional information in comparison to LIDAR point clouds (number of points, first and last return), its accuracy especially in vertical direction, easy availability and lower costs are clear advantages for bigger scale change detection [TerraDrone, 2019].

Two main use cases of point clouds can be found in change detection. First, the point cloud can be used as an additional information input and change detection is mainly done on the images. Often the point clouds are furthermore converted to the DSM to assure the inter-operability [Teo and Shih, 2013; Murakami et al., 1999]. This approach will be adopted for this thesis with converting the photogrammetry-based point cloud into a DSM.

Second, next to the usage as an additional feature, the point clouds are also often used as a pre-selection. Pang et al. [2018] use photogrammetry point clouds to pre-select building areas for change detection using graph-cut algorithms. For a further change detection an algorithm incorporating straight lines from edge detection was developed. This algorithm is promising for this thesis as well, as the straight lines could be a valuable feature.

The other possible approach is to rely more on the points itself. The colour information from the images is projected on the points of the point cloud and information is derived directly from the points. Du et al. [2016] propose an automatic method to detect changes for buildings with aerial images and LIDAR data. Furthermore, an additional photogrammetric point cloud is created from the images and co-registered with the available point cloud to increase the number of points. The dense point cloud from several time points is compared for height and grey-scale difference. The change detection is enhanced using graph cuts, which could be used as an additional feature.

### 3.3 MACHINE LEARNING METHODS

The underlying idea of ML approaches is using example data for changes to extract general rules when a change happens and apply these to new data. In the last years these approaches were continuously improved and are currently the most used approaches for change detection. Many different techniques are available in ML with different goals, so that there is no best to use. Vignesh et al. [2019] compare several ML-techniques especially for change detection. They conclude that deep learning algorithms produce a better accuracy than traditional ML-approaches. However, deep learning approaches need even more datasets for training which are often not available.

The classic way for ML techniques is a foregoing, determined feature collection with following training. Pessoa et al. [2019] are using a random forest approach for change detection using remote sensing images to create high density photogrammetric point clouds. Radiometric and geometric features are collected and used separately and combined for the training of a change detection algorithm. With combined features a very high accuracy of 98% positive detected results could be achieved. Even though the images consisted of an additional NIR-band, which will not be available in this thesis, the collected features and design of the approach gives valuable information for feature extraction, especially regarding the geometric features. To minimize the efforts for feature extraction Han et al. [2019] are using pre-trained feature extraction networks (for example ResNet or AlexNet). These networks are pre-trained to extract information from images and convert into numerical values. Zong et al. [2013] are combining high resolution aerial images and LiDAR data for an improved change detection. They apply three supervised ML methods (Artificial Neural Network, Support Vector Machine and Logitboost algorithm) and compare their results. Their results state that these approaches for change detection can give better results than the traditional methods. Furthermore, they suggest using random sampling to improve the results even more. However, the results should be considered with caution. Only small scenes with many changes were used as an input which does not represent reality in urban structures. Still this paper can be used for performance comparisons with the approach described in this thesis.

A recently developed ML technique especially suited for images is a Convolutional Neural Network (CNN). This algorithm can learn itself what are important features. A good example is the paper of Ji et al. [2019] for change detection of buildings. Next to the change detection process this paper also introduces a method for generating own samples for positive changes, something that could be used for other ML approaches. For this thesis this paper adds less to the actual ML part but contains useful information on how to evaluate the results with scores.

Even though XGBoost is a recently developed library (stable release in 2019), it is already used with success in many applications with aerial imagery, mostly for classification [Georganos et al., 2018; Zhang et al., 2019]. Cao et al. [2019] were exploring classification especially for urban scenery incorporating multi-source geospatial data and different classification approaches. They come to the conclusion that XGBoost performed best due to its attributes for handling overfitting and missing data. Their paper can be used as inspiration for the own workflow of this thesis. Change detection approaches are used less often with the only example found considering change detection for land coverage in the paper of Abdullah et al. [2019]. Similar to this thesis, features from different years were processed and following a classification with XGBoost was applied for change detection. As input data only satellite images were used and the change detection was applied on a larger scale for land coverage. However, the general process stays the same and can give valuable information regarding feature selection and evaluation.

### 3.4 CONCLUSION

As can be seen in table 3.1 the majority of methods for change detection is focusing on a certain type of objects.

| Category             | Buildings | Vegetation | Streets | Water |
|----------------------|-----------|------------|---------|-------|
| Abdullah et al.      |           | X          |         |       |
| Benedek and Sziranyi | X         | X          | X       | X     |
| Cao et al.           | X         | X          |         |       |
| Ji et al.            | X         |            |         |       |
| Knudsen and Olsen    | X         |            |         |       |
| Nebiker et al.       | X         |            |         |       |
| Pang et al.          | X         |            |         |       |
| Rowe and Grewe       | X         |            |         |       |
| Song et al.          |           |            | X       |       |
| Zhou et al.          | X         |            |         |       |
| Zhang et al.         |           |            | X       |       |
| Zong et al.          | X         |            |         |       |
| <b>This Thesis</b>   | X         | X          | X       | X     |

Table 3.1: Overview about the papers and the target objects for change detection

Holistic approaches in which multiple object types are considered are rare. This paper tries to fill this gap using a ML approach.

To have a better overview about the reviewed papers, table 3.1 sorts them accordingly to their input data and the type of their algorithms.

|              |              | Change detection techniques |   |  |                                  |                          |            |
|--------------|--------------|-----------------------------|---|--|----------------------------------|--------------------------|------------|
|              |              | Traditional algorithms      |   | Machine learning   |                                  | Other                    |            |
|              |              | Pixel-based                 | Object-based  | Tree-based   | Deep learning                    |                          |            |
| Data sources | Images       | Remote sensing              | Ilsever and Ünsalan<br>Singh                                    | Ilsever and Ünsalan<br>Peng and Zhang  | Abdullah et al.                  |                          |            |
|              |              | Aerial images               |   | Benedek and Sziranyi<br>Du et al.<br>Knudsen and Olsen<br>Nebiker et al.<br>Pang et al.<br>Rowe and Grewe<br>Song et al<br>Zhang et al | Cao et al.<br><b>This thesis</b> | Ji et al.<br>Zong et al. | Cao et al. |
|              | Point clouds | LIDAR                       |   | Du et al.  |                                  | Zong et al.              |            |
|              |              | Photogrammetry              |   | Du et al.<br>Pang et al.<br>Nebiker et al.   | <b>This thesis</b>               |                          |            |
|              | Digital maps |                             | Knudsen and Olsen<br>Song et al.<br>Zhang et al.<br>Zhou et al. | <b>This thesis</b>   |                                  |                          |            |
| Other        |              |                             |   | Cao et al.   |                                  | Cao et al.               |            |

Figure 3.1: Overview about the papers and their input sources and techniques

Even though there are many different methods with object-based change for almost every input data, the trend is clearly going to the direction of machine learning, as these methods can deliver good results with less effort for data preparation. Findings from change detection from other fields, e.g. change detection in photos or medical images, can be used as an inspiration source for change detection in aerial images.

However, there is no direct paper which can lead as a guideline of implementing or adapting the change detection process using XGBoost. Most ML-approaches today are focussing on deep learning or CNN. However, based on the experiences of [Abdullah et al. \[2019\]](#) and [Cao et al. \[2019\]](#), it is useful to try out XGBoost for change detection combining multiple input sources. Even though it was mainly used for classification until now, applying it to change detection promises interesting results. From all the papers a general procedure of a change detection algorithm can be extracted (not necessary in this order):

- Select the method for change detection.
- Select which object types should be detected.
- Select the suitable input data.
- Select which features are needed.

As a closing remark it was noted during the research, that an up-to-date overview of change detection methods using machine and deep learning methods and their results is still missing.

# 4 | METHODOLOGY

As a conclusion from § 3 it can be said that change detection is usually applied only on certain classes and less in a holistic approach considering multiple classes together. Furthermore, the combination of map and aerial images is not researched yet to full extent. Therefore, in cooperation with Reader, a method was developed to detect changes in aerial images based on a digital map with a use-case of the BGT. In this chapter the complete methodology of the developed change detection will be explained. However, not direct instructions will be given, instead the theoretical background of the methodology will be part of this chapter. Decisions for a certain method or parameters can be found in § 5.4, the results can be found in § 6.

With considering the definitions made in § 2.2 for the learning method, the ML-algorithm used in this thesis can be defined as a supervised classification:

- Supervised: Labeled data is already available, as the target data is already available. Polygons missing this information will be sorted in one of these two categories: changing or not changing.
- Classification: Even though the probabilities assigned are continuous, the general outcome is discrete with the target data consisting of two defined values: changing or not changing.

The structure of this chapter is as follows:

- § 4.1 is describing the workflow of the algorithm.
- § 4.2 is describing the preparation of the data before features can be extracted.
- § 4.3 is describing the extraction of features.
- § 4.4 is describing the actual creation of the model, both tuning of the hyperparameters and the training.
- § 4.5 is describing the evaluation of the results of the model.

The last two paragraphs are not part of the original workflow for change detection. Instead they are explaining how to locate the predicted changes in a polygon (§ 4.6) and give an overview about the possible obstacles that can occur during the project (§ 4.7).

## 4.1 WORKFLOW

The goal is to develop a method based on ML using XGBoost, that can detect changes for digital maps with aerial images and point clouds as input data. In figure 4.1 the workflow for this method can be seen.

The complete workflow can be divided in three large parts:

1. Data preparation  
The input data is cleaned and prepared for the feature extraction.
2. Feature extraction  
The features needed for the ML training are extracted.



### 3. Machine learning

The model for change detection is trained and evaluated. This part is explained in two chapters, as the training and the evaluation of the model take a substantial amount of work.

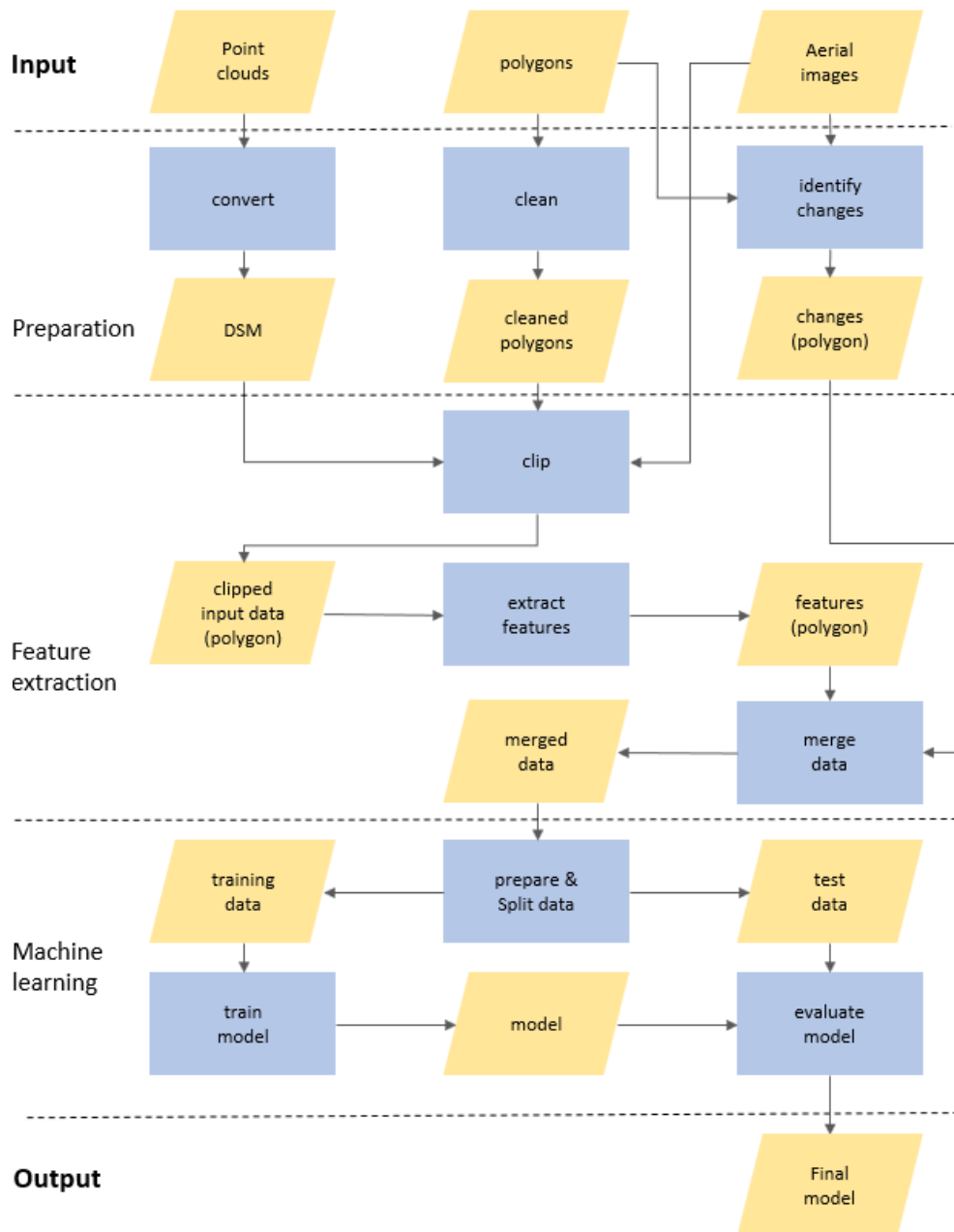


Figure 4.1: Workflow for the change detection algorithm

The workflow is describing the ideal situation, in which the starting features and the first trained model already enable a sufficient change detection. In reality this is usually not the case: Information (and therefore features) may be missing or some features are lowering the quality of the change detection. Parameters and features for the trained model can and must be changed to improve the quality. Therefore, for the last part of ML an iterative workflow is chosen. Figure 4.2 describes this process:

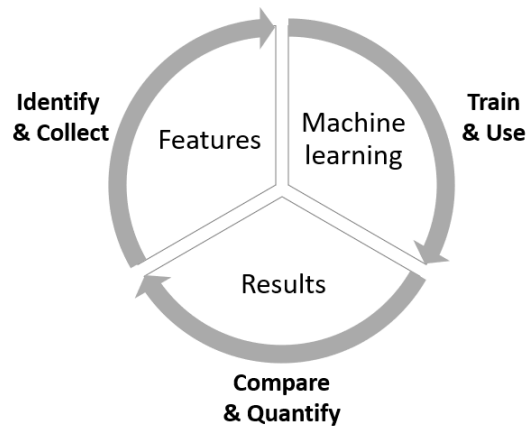


Figure 4.2: Iterative Process for change detection

Whenever a model is trained it is evaluated afterwards. Did the detection rate improve? Are there less false classified changes? The results are quantified not only on change detection itself, but furthermore on the polygons, for example by looking at polygons in QGIS that are false classified and see the values of their features. With this information it can be derived, which information is perhaps missing (therefore which new features must be collected) or which information is misleading. A new model is trained with different parameters and/or features and then is evaluated again.

## 4.2 PREPARATION OF THE DATA

Especially for the training of a ML model the data quality is really important and can determine the success of the complete project. The input data must thus be cleaned before it can be used. It must be assured that all input data - aerial images, point clouds and polygons - are geo-referenced in the same coordinate system, as otherwise the input data cannot be linked to each other.

### 4.2.1 Convert point cloud

When combining a point cloud with aerial images, it is useful to convert the point cloud into a DSM, as then both data sources can be used together more easily: The same programs can be used for visualization and the same algorithms can be used for data extraction.

A DSM can be created out of a point cloud with various methods as described in Zhou [2017]. The regular grid approach is the most commonly used: An empty raster with the extent of the point cloud is created. For each cell all points that are spatially located inside this cell are collected and their average height is taken as the height for the cell. Note that this approach is only working if there is at least one point for every cell, otherwise this cell will contain a Not a Number (NaN)-value. However, this is not a substantial problem, as feature extraction and the ML-model can work with these values.

### 4.2.2 Clean polygons

Depending on the input map, polygons can overlap. As can be seen for example in figure 4.3, the polygons for streets (red) and the polygon for water (blue) are overlapping. However, the polygon for water was drawn at the complete extent of the river and is also located underneath the bridge. When extracting attributes

of the polygons as features, these polygons would disturb the results, as the water polygon would contain attributes of the streets above.

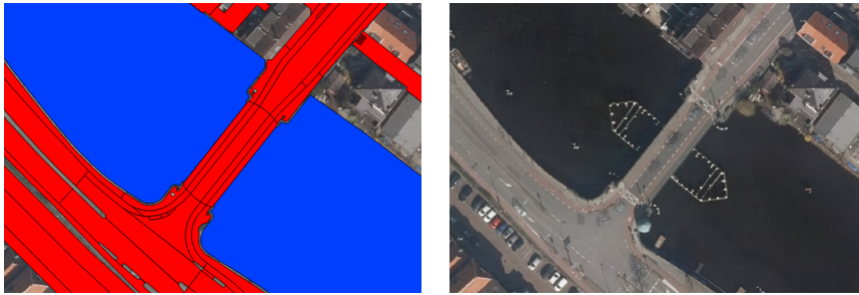


Figure 4.3: Example for overlapping polygons

The solution to this problem is simply to cut the polygons. At every pixel in the research area only one category of polygon should be located. This category must be the category that is visible at the aerial image. For the depicted case in figure 4.3, the single water polygon would be split in a multi polygon, consisting of in total two polygons left and right of the bridge.

Depending on the input data for the polygons some other cleaning must be done as well. Even though the ids for the polygons are usually unique, sometimes they can occur multiple times, as older versions of the same polygon exist and were not deleted. Only the newest polygons should be used and the older polygons with the same id should be removed.

#### 4.2.3 Identify changes

To identify the changes, it is very important to define a change in the context of this change detection beforehand.

##### Definition of change

Change is not a uniform, predefined difference between two states. Depending on the use-case, change can be defined in many different ways. [Chen et al. \[2012\]](#) defines (land cover) change as "variations in the state of physical materials on the earth surface (..)". According to [Bouchaffra et al. \[2015\]](#), three activities can be embedded in the change detection: perceiving the change, the type of change and location of the change. In this thesis two of these three attributes are relevant: Perceiving that a change has occurred and localizing it.

These two mentioned examples already demonstrate that a generally applicable definition of change is difficult. Especially in the context of this thesis, while using a [ML](#) approach with digital maps as the key factor for recognizing changes, there are more things to consider:

1. Administrative or small changes

The [BGT](#) differentiates between classes that only differ in an administrative nature. There are for example separate classes for *wegdeel* and *ondersteunend-wegdeel*. The first class contains main roads whereas the second class contains sections of the road that are less used. However, even for a human it is sometimes difficult to distinguish between both classes. A focus must be set to find differences for both classes even though their visual appearance is similar (for example due to the size or form of the correspondent polygons).

2. Seasonal Changes

Vegetation has a different visual appearance in different seasons. Especially trees can influence the change detection a lot, with visible ground beneath the

trees in the winter and complete green leaves in the summer with no ground visible. Field's appearance also changes throughout the seasons. However, all these changes are no changes in the context of change detection.

### 3. Temporal Changes

Aerial images are snapshots of the daily life in cities. Cars are waiting at red traffic lights, tables are outside of restaurants or there is a market on the main square. All these changes and many more cause a visual difference in the aerial image without being permanent. Hence, they are no changes in the context of change detection.

### 4. Construction Sites

A special case of temporal changes are construction sites. Often it takes more than one year to finish a building and a construction site can be found both years. However, in one year the structure of this construction site can change a lot, especially regarding the height of the constructed buildings. Depending on the exact change, a construction site will be considered as a change from a [ML](#) algorithm (for example construction pit vs. house with some floors already built).

This thesis adopts an own definition of change. This definition can differ from the requirements of changes at the digital map but is necessary to ensure a successful change detection using [ML](#) techniques:

- Everything, that is a clear visual change in the aerial image is considered a change. Even if there is no change in the class (old building is replaced by a new building with a different visual appearance) it is considered a change.
- However, these visual changes must be permanent. Temporal changes or seasonal changes are not defined as a change.
- Construction sites are considered as changes if a change in appearance or a change in height can be found. Otherwise training of the [ML](#) model is more difficult or even impossible.

Usually, a change detected by the algorithm is important to consider for the creators of the digital map, even if the class is not changing. In the example of a replaced building even though the class stays the same, the shape of the new building is likely to be different and the digital map needs to be adapted.

Having a most accurate classification of the existing changes in the input data is very important for reliable results for the [ML](#) algorithm. These polygons are used for training and testing the model and must be correctly classified. As a foundation for changes, the adapted [BGT](#) polygons (see § 4.2.2) of 2017 and 2018 are compared and changes in the object class at the same location are monitored. However, these changes alone are not sufficient. Often there are changes occurring within a [BGT](#)-class or the polygons are not drawn accurately. A manual change detection with a visual examination if a polygon is changing or not must be done as well.

## 4.3 FEATURE EXTRACTION

Even though it would be possible to just input the numeric values of each pixel from the aerial image and the [DSM](#) and apply a separate change detection for every pixel, it is not advised. Object information like the size of the polygon are lost, changes in the colour would already imply a change and it would take too long to evaluate every pixel. Instead feature extraction is used, a "mapping from raw image pixels to discriminative high-dimensional data space" as defined by [Cheng and Han \[2016\]](#).

In this thesis features are gathered for each polygon for each year individually. A polygon is derived from the object from the digital map and usually describes a uniform object like a building or a street. However, in some cases these polygons are drawn in larger scale and can describe a more complex area with different structures located inside (for example the premise of a company containing a parking lot and several buildings). All pixels that are located inside this polygon are used together as an input to extract features. To achieve this, the polygon is depicted on the aerial images and the DSM.

Only the polygons of 2017 (the year of the input map) are known. The polygons for 2018 would be part of the new map and do not exist yet (or at least are not known). To get the features for 2017 and 2018, the aerial images and point clouds from 2017 and 2018 are used, but only the polygons from 2017 are used for the shapes (see figure 4.4). All features therefore are related to the polygons from the same year.

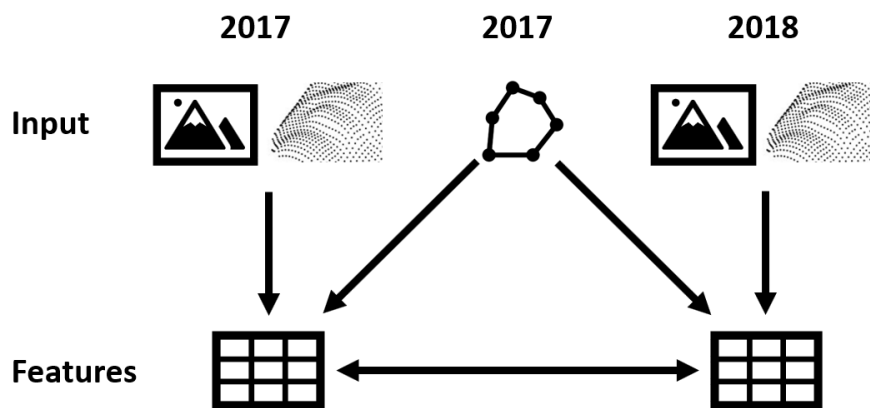


Figure 4.4: Components used in feature extraction

Examples for the actual values of features for a polygon can be found in the annex at § 8.1.

#### 4.3.1 Clipping Input data

Most methods that create features out of images only work with the input data having a rectangular format (either because the algorithm is designed this way or the input data must have the form of an array). However, as can be seen in figure 4.5, most real life objects do not have a rectangular shape. Even if they would have a rectangular shape it is very likely rotated.

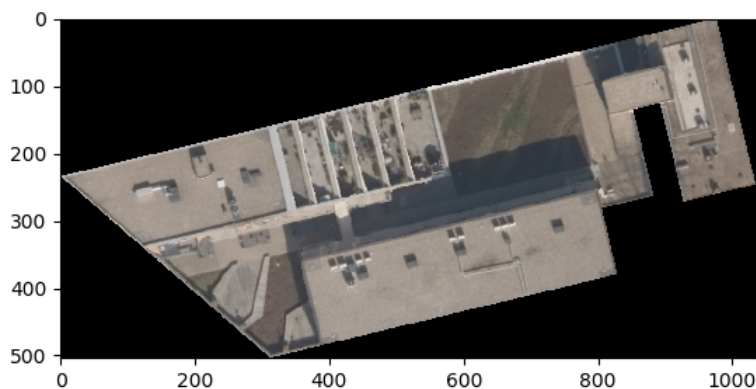


Figure 4.5: Example for a non rectangular polygon

The solution to this problem is clipping and replacing values. In the first step the image is clipped to a rectangular shape, consisting of the minimum and maximum x- and y-values of the polygon. In the second step, all pixels that do not belong to this polygon are replaced with the value NaN. It is a numeric value, however it should express an undefined or non-displayable value. When processing the image, pixels with a NaN-value are ignored.

#### 4.3.2 Calculating features

Depending on the exact feature type different algorithms and methods are applied. In general, the workflow for calculating features is the same for each polygon and both years:

1. get the clipped input data
2. get the numerical values of the pixels
3. apply algorithms on these values and calculate a numeric value
4. store this numerical value in the database

The following features were extracted from the beginning:

- **RGB**: statistical information about the values of red-, green- and blue of the pixels.
- **HSV**: statistical information about the values of hue-, saturation- and value of the pixels.
- **Height**: statistical information about the height pixels.
- **Slope**: statistical information about the slope of a polygon.
- **Aspect**: dominant direction of the slope of a polygon.
- **Shadow**: percentage of the pixels that have a shadow.
- **Shape**: Information about the shape of the polygon.
- **Haralick**: Haralick features for the polygon.
- **Category**: Category of the polygon in 2017.
- **LBP**: Local binary patterns for the polygon.
- **Fourier**: Fourier transform for the polygon.

#### 4.3.3 Merging data

Both datasets from 2017 and 2018 are merged based on their gml-id. This attribute is a unique identifier for every polygon extracted from the BGT. As the same polygons are used for both years it is certain that the same ids can be found in both datasets. Some features are removed in this process, as they are identical for both years and therefore contain unnecessary information. E.g the area of a polygon is calculated in both years, but as the polygons are the same, their size is identical. So only one area-feature is kept, the other is deleted.

Afterwards the changes are added as an additional variable to the data to use them later as the target value. The information for the change is coming from a PostgreSQL table. If an ID is located in this table, the value for this column is 1 (a change has happened), otherwise this value is 0 (no change has happened).

### 4.3.4 Preparation of the features

XGBoost allows only numerical values as an input. However, some of the extracted features are not numerical, the best example is the feature category. Not using it is not an option, as it contains important information for the change detection. In this case the text values must be encoded as numbers in a process called categorical encoding. A categorical value is transformed into one or multiple numerical features. There are multiple options available for this process [Laurae, 2017]:

- **Numeric encoding**  
This is the most basic categorical encoding. Every unique text value is just translated into a unique numerical value. Either an arbitrary number is assigned or a number in order. As this approach only replaces the values, no additional columns are needed. However, with just numeric values a relationship between the classes can be implied (two categories a and b are replaced with values  $a=1$  and  $b=2$ . Now it is implied that b is double the value a). This false information can decrease the quality of the results.
- **One-Hot encoding**  
To overcome the problems of implied false relationships between categories, One-Hot encoding can be used. As can be seen in figure 4.6 for every unique non-numerical value a new column (F1 to F3) is created. All entries that are different from this value have the value 0, entries with the same value have the value 1. There is no relationship between the encoded values, but depending on the number on unique values, many new columns are created.

| Category |    | F1 | F2 | F3 |
|----------|----|----|----|----|
| Building | -> | 1  | 0  | 0  |
| Street   | -> | 0  | 1  | 0  |
| Water    | -> | 0  | 0  | 0  |
| Building | -> | 1  | 0  | 1  |

| Category |    | B1 | B2 |
|----------|----|----|----|
| Building | -> | 0  | 1  |
| Street   | -> | 1  | 0  |
| Water    | -> | 1  | 1  |
| Building | -> | 0  | 1  |

Figure 4.6: Comparison of One-Hot encoding (left) and binary encoding (right)

- **Binary encoding**  
Similar to One-Hot encoding new columns are created to overcome the problem of implied relationships. However, in this method binary information is used to encode these categories which results in less columns being needed. Instead of using a new column for every unique value, only as many columns as bits are needed to distinguish between the unique values. For example with eight different values, three columns would be needed, as three bits are needed to describe the number eight. As shown in figure 4.6, the number of columns is already reduced by one.

## 4.4 MODEL TRAINING

In this section the training of the model of the XGBoost algorithm is described. As shown in figure 4.7 the actual training is only a part of the whole procedure. Equally important is the right split of the data and a solid selection of the hyper parameters.

When training a binary classification model, two terms are important to know, as they are the two general types of errors influencing the quality of the model [Yao, 2019].

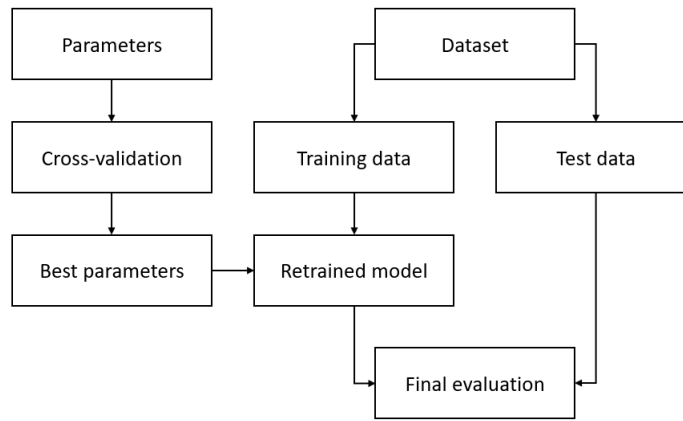


Figure 4.7: Workflow for tuning and training of the model (© scikit-learn)

- Bias is the overall difference between expected predictions made by the model and true values.
- Variance describes how much predictions for the given point vary.

The ultimate goal is to keep both errors small. Training a model always means finding a balance between these two factors. The model should not predict too few good results (underfitting) but also not create a model predicting all results correctly just for the training data (overfitting).

#### 4.4.1 Splitting the data

As a first step the data must be split. A small part of the data should be put away and never be considered during the training. This set is the test data. As can be seen in figure 4.8, the complete training is done only with the training data. If the complete data would be used for training, the following evaluations would be distorted. Instead of predicting new values, the results are directly derived from the training and the model would perform great. But as soon as completely new data must be evaluated, the model fails. The model is overfitting.

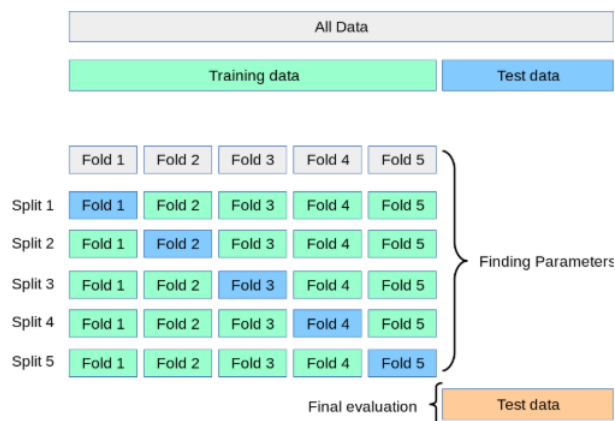


Figure 4.8: Example for k-cross folding with  $k = 5$  (© scikit-learn)

There are multiple options for the splitting of the data. First the ratio between training and test data must be decided. In both sets there must be positive target values. Second, it must be decided on how to split the data:



1. It is possible to shuffle the data and then randomly select a percentage of the data as the test data. However, with a very imbalanced dataset it would be possible that there are positive values available in the test-set.
2. The percentage of test-data can also be applied individually for positive and negative values. A certain percentage of all entries with positive and the same percentage for all entries with negative values are selected as test data. It is therefore certain that there are always positive values available in the test-set.
3. Both of those methods use randomly selected data. However, sometimes it is important to also consider the spatial relationship between polygons. It was noticed that changes are usually clustered: In most cases multiple polygons located next to each other all are changing and it is more rare that only a single polygon is changing. It can be valuable to have polygons located next to each other in the same test- or training set. An option would be to base the selection of data on the tiles used for the change detection (75% of the tiles as a training-set and 25% as a test-set)

Depending on the exact training method used, the training data also must be split again in the actual training data and a smaller set of validation data.

#### 4.4.2 Model parameters

As described in § 2.2, many different models are available for ML as well as for XGBoost. Some basic decisions for the model must be taken beforehand as they are the foundation for the whole operation. The following general parameters are important:

- **Booster**  
Decides which booster should be used. A booster is a kind of a framework and defines the methods and parameter that can be used. As the whole algorithm should be tree-based, `gbtree` is the only possible selection.
- **Objective**  
Specify the learning task of the algorithm. This value is depending of the desired result of the model. For binary classification the most common selection is `binary:logistic`.
- **Eval\_metric**  
The evaluation metric defines how the quality of the model is measured. Depending on the dataset itself and the desired results, different parameters can be used here. As these parameters are exactly the same as in the later evaluation of the whole model, they are described in § 4.5.2.
- **Scale\_pos\_weight**  
With a very unbalanced dataset, such as the the one used for this thesis, this parameter can set the focus on the positive samples and helps in faster convergence of the model to improve the results. Usually the weight is calculated with formula 4.1:

$$scale\_pos\_weight = \frac{\text{number of negative target values}}{\text{number of positive target values}} \quad (4.1)$$

However, with a very imbalanced dataset this parameter can get very large values and can be calculated instead with formula 4.2.

$$scale\_pos\_weight = \sqrt{\frac{\text{number of negative target values}}{\text{number of positive target values}}} \quad (4.2)$$

There are some parameters that are not important for change detection itself, but are useful to adapt:

- **Verbosity**  
This parameter controls the printing of messages. In default mode only error messages are printed. However, especially during development it can be helpful to see exactly how many trees are used and the exact attributes of these trees.
- **N\_thread**  
The number of different threads (virtual cores) that are used by the algorithm. By default, the maximum number is used. However, sometimes this behaviour is not desired and less threads can be set with this parameter.

Learning task parameters specify the learning task and correspond to the actual learning objective. There are more parameters available, however these are usually less important and have a very small influence on the end-results. The following parameters were adapted:

- **Learning\_rate (eta)**  
The learning rate describes the influence of a new decision tree. A smaller learning rate prevents overfitting and makes the model more conservative.
- **N\_estimators**  
The number of gradient boosted trees and equivalent to the number of boosting rounds. More trees allow more complex models but are also more likely to overfit.
- **Max\_depth**  
With the max\_depth the maximum depth of a tree can be set. Increasing this number means that the trees can be more complex and possible interactions between features can be recognized. However, it is also more susceptible to overfitting. This setting only describes the maximum allowed depth, therefore trees can still have a smaller depth (if they are pruned for example).
- **Min\_child\_weight**  
Defines the minimum sum of weights of all observations required in a leaf. It is used to control overfitting, as higher values prevent learning very specific relationships between features.
- **Colsample\_bytree**  
The number of features that are randomly sampled when building a tree. Instead of using all attributes for a tree, each time only a subset is used, so that every tree is built from different features. This reduces overfitting of the model.

#### 4.4.3 Parameter tuning

There are many different ways to tune the hyper parameters<sup>1</sup>. Usually the parameters are discretized beforehand with a list of possible values [Restrepo, 2019]. The number of possible combinations can rise quickly, with every parameter having six values. With five parameters there would be  $6^5 = 7.776$  possible combinations. The following two methods are most commonly used:

- **Grid Search**  
If there is a smaller number of parameters that are tuned Grid Search can be applied. All combinations of parameters are tried and evaluated. This method will give the best combination of parameters, however it can take a long calculation time with many possible combinations.

<sup>1</sup> see [Wikipedia](#) for a full list.

- **Random approach**  
In this approach a random value is selected for each parameter. The advantage is its easy implementation and it is likely (with a sufficient number of rounds) to come close to optimal parameters. However, there is a high chance that often the search focuses on uninteresting parts and not every combination is tried out. This approach is selected if there is a high number of parameters to tune, as not every combination is selected.

A good way to avoid overfitting and increase the quality of the hyper-parameter tuning is cross validation as depicted in figure 4.8. In cross validation the whole test-data set is split in  $k$  multiple 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. The performance measure of a  $k$ -fold cross validation is the average of the validations in the loop. The disadvantage is that the number of models needed to be created and evaluated is  $k$ -times bigger. When using cross-validation no validation data is needed.

Another method to prevent overfitting is the usage of early stopping for getting `n_estimators` [Brownlee, 2019a]. The performance of the model is monitored with looking at the test-set. If the performance of this set is not improving over a certain number of rounds or is even decreasing (due to overfitting), the training is stopped. The time-step in which the data is not improving any longer is the new number for `n_estimators`.

#### 4.4.4 Training & applying the model

Change detection can be considered as a binary classification, with having the change as True- and no change as a False-value. When training the model, XGBoost builds a model considering the previously determined hyper-parameters. Like described in § 2.3.3, features are randomly selected to build the best-fitting decision trees. These trees together form the model. The training itself is done in background by calling a function and the created model can be saved as file for later use. All data from the training-set is used to train the model.

## 4.5 MODEL EVALUATION

In order to evaluate the model, the test-set is applied to the trained model from § 4.4. Changes are predicted and can then be compared with the real changes. Contrary to the training, it is very important to only use the features and never the target values. Two options are available for binary classification:

- `predict`  
This option will just return the most likely class, in the case of change detection either 0 (no change) or 1 (change).
- `predict_proba`  
This option will return both probabilities for the two classes (0 first and 1 as second probability). The sum of both values is always 1.

Multiple insights can be extracted from the model. This includes the importance of the features of the models as well as the results of the model during tuning and the results for the final evaluation set.

### 4.5.1 Feature evaluation

Feature evaluation means that the features used in the model are viewed individually. Important and less important features can be recognized and a selection of final features for the model can be done.

#### Feature importance

Feature importance is already implemented in the XGBoost package. After a successful training all features can be displayed ranked based on different criteria.

Standard evaluation criteria offered by XGBoost are:

- **Weight**  
The number of times a feature occurs in a tree of the model. The more often a feature occurs in a tree, the more often it is used to split the data.
- **Gain**  
Relative contribution of a feature to the model and describes the improvement in accuracy brought by this feature. For every tree this gain is difference and the average value is calculated. The higher the value, the more important this feature is for the model.
- **Total gain**  
The same as gain, but instead of calculating the average gain the sum of all gains is returned.
- **Coverage**  
Relative number of observations related to this feature. Whenever a decision is made in this tree for an end node using this feature, its coverage is increasing. Like gain this value is different for every tree and the average value is calculated. The higher the value, the more often it is used for end nodes. The values are given as a percentage for all features.
- **Total coverage**  
The same as coverage, but instead of calculating the average coverage the sum of all coverages is returned.

#### SHAP

Shapely Additive exPlanations is a library for python developed by [Lundberg and Lee \[2017\]](#) and uses a game theoretic approach to explain the output of ML models. It allows to see directly the contributions of features on a global but also individual scale for every polygon. Furthermore, it can plot how features are correlated with other features. An example for feature evaluation with SHAP can be seen in the paper of [Parsa et al. \[2020\]](#). Several different predictions are possible, a selection of useful predictions for tree-based models is presented in this paragraph.

To explain the predictions of tree-based algorithms, special plots are available based on a paper from the same authors with special focus on these algorithms [[Lundberg et al., 2020](#)]:

Individual predictions describe the feature importance for a single entry. The `forcePlot` shows the features and how much they are contributing to push the model output for a particular entry away from the base value, e.g. as a base value there is a probability of 0.5 that a polygon has changed and the feature "area" lowers the probability by 0.2.

Global predictions are predictions on the features based on all entries together. The `summaryPlot` displays an overview of all used features. Each feature is plotted with all entries and their respective feature values. The `dependence plot` instead is plotting two features against each other in order to recognize correlations between features.

SHAP is used as an additional source for evaluating the feature importance. In comparison to the built-in tools from XGBoost, it is focussing on the feature values of the entries itself.

#### 4.5.2 Result evaluation

Instead of looking at single features, the complete model is evaluated to check how successful the model can classify unknown results. It is therefore also a measurement of how useful the model will be for the future task. These evaluations are used at many opportunities, including the hyper-parameter tuning, for model selection and for the final evaluation of the project success. However, there are many different evaluation models all with a focus on different parameters.

A general problem of all evaluations is the basic definition of what a successful model is. Is it more important to get all True-values (maybe even with a cost of many false classified values) or is it more important to have less False-values? This decision is really dependent on the use-case the model is applied for.

#### Confusion matrix

A classic approach often used for binary classification in ML is the Confusion Matrix (CM). It is a table with a specific layout in which the results of the model can be displayed. An example for the layout is depicted in table 4.1.

|            |       | Change   |          |
|------------|-------|----------|----------|
|            |       | Positive | Negative |
| Prediction | True  | TP       | TN       |
|            | False | FP       | FN       |

Table 4.1: Example for Confusion matrix

In this matrix the number of predicted target values (changes) are compared with the number of actual target values. Following four combinations are possible (based on this project):

- True Positive (TP): Real change and change detected
- True Negative (TN): No real change and no change detected
- False Positive (FP): No real change and change detected
- False Negative (FN): Real change and no change detected

The advantage of the CM is the easy visualization of the performance of the model. The performance is not displayed as an abstract number but with values that also people with less technical background can understand. However, there are two problems. First, to distinguish between True or False a threshold must be set when a probability of a change is considered as True or False. Depending on this threshold the performance of the model can differ a lot. Second, multiple CM from different models are difficult to compare:

- Which values should be ranked higher? Less FP or more TP?
- With a different number of input entries in different confusion matrices it is difficult to compare them directly.
- Matrices with different thresholds cannot be compared.

A threshold is always a value between 0 and 1 and converts the probability of a polygon into the two classes change and no change. Every polygon with a probability smaller than the threshold is considered as no change, every polygon with a probability equal or higher is considered as a change. For the threshold usually default values like 0.1, 0.5 or 0.9 are used. With lower thresholds more changes can

be detected with the cost of having more non changes detected as changes. With a higher threshold less changes are detected but also less non changes. Another approach is to calculate the threshold mathematically<sup>2</sup>. For this approach it is tried to find a point in which the combined value of precision and recall is the maximum.

### Curves

Curves are a visual description of the model and can bypass the problem of setting an initial threshold. Instead of having one threshold, all possible threshold values from 0 to 1 are displayed. It is important to know that the threshold values are not displayed on these curves. These values are implied with the curve itself. Different curves are available that can highlight different attributes of the model [Brownlee, 2018]:

- 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 first on the y-axis and the former on the x-axis. The TPR tells what proportion of positive labeled entries are correctly labeled, the FPR tells the proportion of negative labeled entries that were incorrectly labeled. At the top right corner everything is labeled as positive, at the bottom left corner everything is labeled negative. The probability for a change is 0.0 on the left side of the curve, whereas it is 1.0 is on the right side.

- Specificity

This value, also known as True negative rate (TNR), is part of the ROC-curve. It is calculated using formula 4.3.

$$\text{specificity} = \frac{tn}{tn + fp} \quad (4.3)$$

- FPR

The FPR is another part of the ROC-curve. It is calculated using formula 4.4.

$$FPR = \frac{fp}{fp + tn} \quad (4.4)$$

The more the curve tends to the upper left corner, the better the model makes correct predictions. An ideal curve goes straight up and then horizontal to the right. When using a random assigner, the curve would be a straight line from the bottom left to the top right. Every model that has a line below this curve is a worse classifier than using a random approach.

- Precision recall curve

As the name suggests, the Precision-Recall (PR)-curve plots the precision of a classifier with the recall of a classifier. Depending on the threshold these values are changing. These curves are more commonly used with imbalanced datasets where the few examples are positive, as the high number of FP are not displayed. For this curve only the correct prediction of the true target values are important.

The more the curve tends to the upper right corner, the better the model classifies the positive results correctly.

---

<sup>2</sup> see [here](#).

- Economic curve

The economic curve is a curve used by Readar for many projects. It describes the number of entries that must be checked manually to assure a certain percentage of found changes. The more the the curve tends to the upper left corner, the more economic is the model.

With balanced datasets it is better to use the ROC-curve, whereas the PR-curve is better used with imbalanced datasets. For imbalanced datasets the ROC-curve gives too optimistic views so that the performance of the model is overestimated [Brownlee, 2018].

### Scores

Scores are defined in this thesis as evaluation criteria consisting of a single number that can describe the output of a model. There are two main sources from which these numbers can be derived: The CM and from the curves of the model.

Like the CM, the derived scores are dependent on the threshold value. There are more scores available, however the presented scores are the most common ones used in a ML context [Ferreira, 2018; Gunawardana and Shani, 2009].

- Accuracy

One of the best-known evaluation scores both in binary classification as well as in ML is the accuracy. It is calculated using formula 4.5 and is the fraction of right classified polygons.

$$accuracy = \frac{tp + fp}{tp + fp + tn + fn} \quad (4.5)$$

- Precision

Precision gives an indication how many of the positive predicted results are correctly classified and is calculated using formula 4.6. It is an important indicator to consider if the number of FP should be reduced.

$$precision = \frac{tp}{tp + fp} \quad (4.6)$$

- Recall

This value, also known as sensitivity, gives the percentage of positive changes that were recognized correctly. It is calculated using formula 4.7.

$$recall = \frac{tp}{tp + fn} \quad (4.7)$$

- F1 Score

The F1 score also measures the accuracy, however only the positive results are important. It is calculated using formula 4.8.

$$F1Score = 2 * \frac{precision * recall}{precision + recall} \quad (4.8)$$

There are also parameters that are derived from curves and are therefore independent from setting a certain threshold.

- **AUC**  
The Area under curve (AUC) is used to describe a curve in a single number. It describes how much area of a plot is located under the curve and is used as a method to compare different models. To assure comparability between different plots this number is expressed as the percentage of the total area of a plot. Even though AUC is the general name for this technique, it is usually referred to the ROC-curve.
- **PR-AUC**  
Similar to the normal AUC, this parameter describes the area under curve, however in this case it is referred to the AUC from the PR-curve.

## 4.6 LOCATING CHANGES IN POLYGONS

Many polygons only cover a small area and if a change is detected, usually the change is occurring in the whole extent of the polygon. However, for some polygons (either because they have a large area and/or a very irregular shape) only a part of the polygon is changing and localizing this change visually in this polygon is more difficult.

A possible solution is to split polygons in smaller units and apply a change detection on these units. The change detection will therefore be refined and only the polygons that contain changes are marked. Then it is easier to localize the changes in large polygons. Splitting the polygons can be done in different ways:

- **Geometric splitting**  
The simpler method is splitting the polygon in smaller polygons just based on geometric algorithms. This can include using a grid, triangulation or methods that split a polygon in sub-parts of equal size and shape.
- **Image segmentation**  
A more progressive method would use image segmentation methods to split the polygons based on certain attributes. All parts where this attribute is similar form one unit. This can be done based on colour, height or both.

A positive side effect next to the localization of the changes is that the results of the model are improved. Many errors in the classification are caused by polygons in which only a small subset is changing. Due to its small extent, this change is not depicted within the features of this polygon. The statistical values are less influenced. In the smaller subsets of this polygon it is more likely to have polygons, in which full extent of the area is changing. This is depicted better within the features and it is more likely to detect this change.

## 4.7 POSSIBLE OBSTACLES

In the following section obstacles for this project and their mitigations are listed.

- **Temporal differences in the aerial images**  
As visible in figure 4.9 the input aerial images can be very different depending on the recording time. Vegetation in spring is most likely very green and prevent the view to objects beneath it, whereas in winter it is more brown and objects beneath can be seen. Therefore, the colour features of some polygons will be very different and hard to classify using ML. A possible solution to this



problem would be to rely less on the colour information and more on features reluctant to temporal changes.



Figure 4.9: Same scene in 2017 (left) and 2018 (right)

- Variation inside the classes  
The same classes can have a big variation and therefore a different shaping with multiple colours or textures. Again figure 4.9 is a good example for this obstacle. This street has a lot of deviations (trees) that will influence especially the RGB values for this polygon. Other streets do not have these deviations. Still both need to be correctly identified as streets. Not only the colour information but also texture information must be considered with caution here. More successful is in this case a focus on the shape of the polygon (a street is long but thin).
- Imbalanced number of classes  
Cities undergo constant changes, even though compared to the total size of the city the number of changes are small (for the research area of Haarlem there are around 150.000 polygons and 1.400 changes, around 1%). This imbalance makes it difficult to get a good change detection using ML. A possible solution would be to increase the importance of detecting true changes. Furthermore, it would be possible to increase the number of training data with changes, so that a balance between changes and non-changes is assured.
- Small proportion of change  
Sometimes only a small part of the polygon is actually changing. A good example would be a huge field on which a small house is built. Most of the polygon is still a field and only a little bit has changed. Many features (like average) will only be slightly changed and a detection of this change is difficult. Features must be found that change significantly even if only small changes occur (and are furthermore robust to temporal changes).

# 5 | IMPLEMENTATION

In this chapter the exact implementation of the algorithm is described. This includes a specification of the input data and the decisions made during the execution of the project. Afterwards the results and computation times are displayed and compared with other methods.

## 5.1 RESEARCH AREA

The research area for this master thesis is the municipality of Haarlem, a city located northwest of Amsterdam in the Netherlands (see figure 5.1). All data and therefore the training and validation of the ML-algorithm is limited to the boundaries of the city.



Figure 5.1: Location of Haarlem (© Wikipedia)

The city was selected to be the study area, as it has a good BGT-quality and aerial imagery in high resolution is available. Both digital map and aerial images are available for the consecutive years of 2017 and 2018. With this imagery a point cloud could be created via photogrammetry. Furthermore, a good selection of different surface textures can be found, among them a traditional inner city core with small houses, residential areas with gardens, business and industrial areas, waterways and even some agricultural area.

## 5.2 INPUT DATA

In the following sections the input datasets of the BGT (§ 5.2.1), the aerial images (§ 5.2.2) and the point cloud (§ 5.2.3) are specified.

### 5.2.1 BGT

The polygons from the digital map are available in a structured format. Each row is one unique polygon and consist of a unique ID (the gml-id), the class and its exact location and shape. The data is available for both 2017 and 2018.

The classes of the **BGT** are used as a category for the change detection. However, not all classes are considered in this thesis for the following reasons:

- For some classes the small sample size is too small and too little data is available to train a model for these particular classes.
- Some classes (for example kunstwerkeel - additional infrastructure resources) consist only of polygons with a very small shape and the resolution of the aerial images is not high enough to extract meaningful information.
- Some classes are not visible on the aerial images and no information can be extracted. Examples include tunneldeel (tunnel structures under the ground) or overbruggingsdeel (parts of street that are on bridges; looking exactly like the class wegdeel)

The classes used for change detection in this master thesis can be seen in figure 5.2.

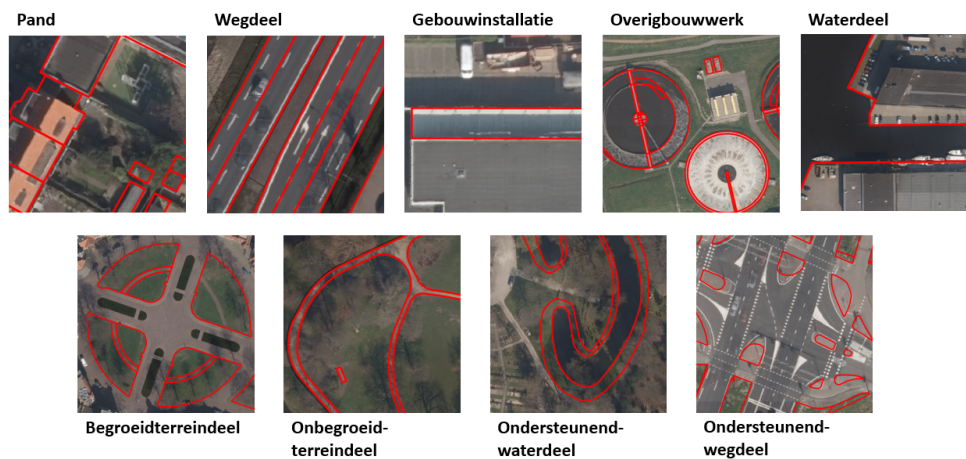


Figure 5.2: Used BGT class in this thesis (marked with red lines)

Table 5.1 gives an detailed overview about the distribution of the objects. Even without the other possible classes, the input **BGT** data covers over 99% of the city area of Haarlem with a total number of 161.762 objects.

| Class                  | Description            | Number of Objects | coverage ( $km^2$ ) |
|------------------------|------------------------|-------------------|---------------------|
| Pand                   | Buildings              | 81.104            | 5.715               |
| Wegdeel                | Streets & ways         | 32.742            | 6.381               |
| Gebouwinstallatie      | Building installations | 1.536             | 0.018               |
| Overigbouwwerk         | Other buildings        | 1.495             | 0.078               |
| Waterdeel              | Waterways              | 2.048             | 4.681               |
| Begroeidterreindeel    | Vegetation             | 12.777            | 8.533               |
| Onbegroeidterreindeel  | Bare ground            | 19.894            | 6.751               |
| Ondersteunendwaterdeel | Supporting waterways   | 3.050             | 0.446               |
| Ondersteunendwegdeel   | Supporting streets     | 7.116             | 0.935               |

Table 5.1: Amount and coverage of BGT-objects

The data for the **BGT** is available on websites of the Dutch Government <sup>1</sup>.

<sup>1</sup> see [here](#).

### 5.2.2 Aerial images

The aerial images used in this thesis are part of the stereo10-dataset. This dataset is provided by the company Cyclomedia and contains true-ortho imagery for the complete Netherlands. It is offered on an annual basis and has a resolution of 10 cm. To assure its usability for digital processing, the images are usually taken in the leafless season of early spring and late autumn and with no present disturbances like clouds. A maximum visibility is therefore assured.

Images for both 2017 and 2018 are used as input data. They are available in the TIFF-format and consist of three bands (RGB). However, these images were captured in April and thus have already some green vegetation.

### 5.2.3 Point cloud

The point cloud is created with 3D photogrammetry from the aerial imagery used in this thesis. It was created by the company using an internally developed approach. Complementary to the default algorithms used for 3D photogrammetry a ML technique is used to improve the results.

As the point cloud is created by photogrammetry, the number of points is equally distributed for the different surfaces. 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.

### 5.2.4 Changes

Based on the criteria defined in § 4.2.3 the changes were detected manually. In order to ensure a maximum accuracy with no polygons left unchecked, the whole research area was split in tiles of 250x250m and each tile checked separately. In total, there are 1.378 polygons that are changing between 2017 and 2018. Compared to the total number of polygons, only 0.85% of the polygons are changing. Table 5.2 shows the exact distribution of changes.

| Class                  | Changes (% of class) |
|------------------------|----------------------|
| Pand                   | 464 (0,57)           |
| Wegdeel                | 323 (0,99)           |
| Gebouwinstallatie      | 4 (0,26)             |
| Overigbouwwerk         | 4 (0,27)             |
| Waterdeel              | 6 (0,29)             |
| Begroeidterreindeel    | 271 (2,12)           |
| Onbegroeidterreindeel  | 186 (0,03)           |
| Ondersteunendwaterdeel | 6 (0,2)              |
| Ondersteunendwegdeel   | 114 (1,6)            |

Table 5.2: Changes per class

## 5.3 TOOLS

The prototype software was mainly created in Python with additional Matlab scripts called via an API. Data storage and management was done in PostgreSQL with the PostGIS extension. For data preparation, visualization and evaluation QGIS was used. In the following subsections these components are described.

### 5.3.1 PostgreSQL

Data storage is done using the database management system PostgreSQL with PostGIS extension for spatial data. Figure 5.3 shows the structure of the database.

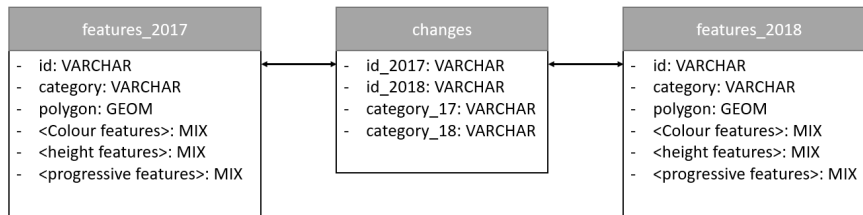


Figure 5.3: Database scheme for the features

The two feature tables (features\_2017 and features\_2018) contain all the collected features for the polygons respective for the separate years. The unique id (derived from the gml\_id) is the primary key. The column polygon contains the exact shape of the polygon. For both years the id and the shape are from 2017, as the polygons from 2018 are not known. As one feature table contains around 160.000 entries and to enable a smooth selection via geometry, the polygon column has a spatial index. The table changes is linked with the gml\_id from 2017 to both tables features\_2017 and features\_2018. An ID is only located in the table changes if the belonging polygon is changing.

Three tables store all the data needed for the change detection. Note that only a simplified schema is shown in figure 5.3, in reality the table for the features consist of more columns (one column for each collected feature). The actual input data itself is not stored in the database but as files.

### 5.3.2 Python

Python is the main language used in this application with all major tasks developed as python applications. The advantages of python are its easy readability, easy-to-use environments (both developing and executing) and its extensive number of libraries. The following libraries are used (this list is not complete but a selection of the most important libraries):

- NumPy  
NumPy<sup>2</sup> adds support for multi-dimensional arrays and matrices along with mathematical operations. It is mainly used for calculating features.
- Pandas  
Similar to NumPy, Pandas<sup>3</sup> allows to handle multi-dimensional arrays and matrices. The focus is less on mathematical operations and more on the data storage. It supports headers for the arrays and a mixing of numeric and text values (in comparison to NumPy). It is used for the whole data storage and for the communication with XGBoost.
- SciPy  
SciPy<sup>4</sup> is a library used for scientific computing. It adds support for many image recognition features that are used for change detection. Furthermore, it enables additional settings and evaluations for XGBoost.

<sup>2</sup> see website of [Numpy](#).

<sup>3</sup> see website of [Pandas](#).

<sup>4</sup> see website of [Scipy](#).

- Scikit-learn  
Scikit-learn<sup>5</sup> is a library specially designed for ML. It adds many algorithms for classification, regression, etc. as well as expanded functionalities for XGBoost.

### 5.3.3 XGBoost

Even though XGBoost is used in this thesis as a python plugin, due to its importance it will be mentioned separately. Next to python, XGBoost is available in more different languages (like R, C, Java, etc..).

There are multiple ways to access the XGBoost objects and functions, the most common method and the one used in this thesis is accessing it with the integration of Scikit-learn. To use the algorithm, a new XGBoost-object must be created, whereas the exact statements are different for the categories mentioned in § 2.2.2.

With these objects all required tasks can be executed: Tuning, training and predicting the target values. These models can be saved as files, allowing a later usage. For more information about the syntax and how to use this library it is referred to [Brownlee \[2016b\]](#).

### 5.3.4 Matlab

Matlab is used for two special algorithms that are available in this environment: Calculation of the shadow and the Fourier transform. Only the Matlab-engine and the files containing the code need to be available, the algorithms can be called via python and processed further.

### 5.3.5 QGIS

QGIS is an open-source geo-information-system and in this thesis mainly used for visualizing and evaluation of the geo-data. It is possible to display the polygons, aerial images and DSM in order to get own visual impressions of the data. Additionally, the detected changes can be displayed here. The manual change detection and the splitting of the polygons is done using QGIS.

## 5.4 DESIGN DECISIONS

In this section, the particular design decisions made during the implementation are explained. The exact numbers and parameters can be found as well as decisions for certain methods. It is separated in following paragraphs:

- Data preparation (§ 5.4.1)
- Feature extraction (§ 5.4.2)
- Model training (§ 5.4.3)
- Model evaluation (§ 5.4.4)
- Locating changes (§ 5.4.5)

---

<sup>5</sup> see website of [Scikit](#).

### 5.4.1 Data preparation

The grid approach is used to create the DSM. A self written python application is used to convert the point cloud. 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.

As there are many overlapping polygons in the BGT (see § 4.2.2), the polygons were overlaid and split if necessary. The split is done based on a ranking, in which the higher ranked polygon is kept and the lower ranked is cut if polygons are overlapping. If the cutted polygon consists of multiple polygons afterwards it is converted into a multi-polygon. Categories are ranked in the following order with consideration of which polygons are usually visible in comparison to each other.

1. gebouwinstallatie
2. pand
3. overigbouwerk
4. begroeidterreindeel
5. onbegroeidterreindeel
6. wegdeel
7. waterdeel
8. ondersteunendwegdeel
9. ondersteundendwaterdeel

To support the manual change detection, the whole research area was tiled temporary in squares of 250x250m and each tile was checked individually for existing changes. The rules described in § 4.2.3 were applied. Polygons with a surface area of less than 2m<sup>2</sup> are not considered for change detection. Their size is too small to extract meaningful information and ensure a successful classification of changes.

### 5.4.2 Feature extraction

Even though many features can be extracted relatively simply from the input data, for some features conversion of the data or other procedures are required. This section describes the decisions made during extraction of the features:

- One feature in the dataset, precisely the category of the polygon, is not numerical but in text-format. Even though One-Hot-Encoding or binary encoding are technically the correct methods, simple numeric encoding was selected as according to tests made by [Laurae \[2017\]](#), the differences in quality are negligible with data having a small number of cardinalities.
- For the feature extraction using Fourier transformation and shadow calculation, algorithms were already developed for another project of the company and the existing Matlab-scripts could be reused.
- For the Haralick features and the LBP a python package (Mahotas<sup>6</sup>) for computer vision was used. For more information for Mahotas it is referred to [Coelho \[2013\]](#).

---

<sup>6</sup> see website of [Mahotas](#).

During training and evaluating the intermediate models using the test-data, some features were removed or new features were tested and included (if they improved the results). The final results can be found in § 6. The following new features improved the results and were included (some features did not improve the results and were discarded; their description can be found in § 5.4.2):

- **Bhattacharyya distance**

XGBoost can also be used for multi-label classification, meaning that instead of two probabilities for True and False it is possible to calculate the probabilities for a polygon to belong in multiple classes. A simple model was created (with tuning and training exactly like in the change detection and using the same features) for classification. For each polygon and each year a probability distribution was calculated. It consists of a probability for every BGT-class of how likely this polygon belongs to this class.

The Bhattacharyya distance describes a property used in statistics and measures the similarity of two different probability distributions. It is used to convert the two distributions of both years into a single number that expresses how similar the two probability distributions are. This is synonymous to the probability of a polygon having the same class in both years. This value turned out to be an important factor for the change detection, even though the same features are used for both multi-label classification and change detection. Inspiration for using the Bhattacharyya distance and more information about this feature can be found in [Choi and Lee \[2003\]](#).

- **Canny edge**

Canny edge detection describes an algorithm used to detect edges in images. With this algorithm it is possible to get information about how homogeneous an image is, as more edges imply more situations in the polygon in which sharp changes in the brightness, surface material or depth (=height) are happening. It is successfully used for classification in both XGBoost [[He et al., 2017](#)] or other approaches [[Huo et al., 2020](#)]. In this thesis canny edge is converted into a numeric value by counting the percentage of edge-pixels in a polygon.

- **nPix\_median\_height**

A problem especially for change detection involving streets are the different height levels caused by cars. A street without cars is completely flat, but with cars there are different height levels. A solution to this problem is this feature. The percentage of pixels that are in maximum 2m distance to the median of the height is counted. This can tell if the polygon is consisting of more flat areas or does have spikes in both directions up and down (these cancel each other out and the median is still similar to flat areas). With the difference of 2m most cars are still counted as close to the median. Empty and crowded streets therefore have a similar value for this score.

- **Differences**

After the unsuccessful attempt to use only the differences as input for the model (see [here](#)), the differences were included as an additional feature. This could improve the model strongly, especially in reducing the number of FP. The differences allow a direct inclusion of the information how high the change between two features is instead of deducing it from the two values (As an example height in 2017 was 20m and height in 2018 was 10m: With a tree-based model this information of the difference of -10m can only be extracted if both values are in the tree).



However, three features were removed, as they influenced the results notable negatively: `height_average`, `height_minimum`, `height_maximum`. Due to errors in the point cloud (see in § 6.5 the point "Usage of percentiles") these features contained data that did not represent the real situation and would lead the algorithm astray.

### Other attempts

More attempts with different constellations of features were examined. These attempts were unsuccessful, as they could not improve the results of the model. However, for completeness these attempts should be shortly described in this section. As these attempts were not part of the workflow the results are not displayed in chapter § 6. However, for interested readers the results can be found instead in the annex at § 8.2.

- **No Colours**  
Many temporal changes in the images are caused by changes in the visual appearance (especially changes in the vegetation due to the seasons or surface changes like different concrete). It was checked if all features excluding only `RGB` based features in the first attempt and excluding `RGB` and `HSV` based features in the second attempt can still deliver good or even better results.
- **Only difference**  
Instead of taking the features for 2017 and 2018 separately, the differences between both values are taken where applicable (e.g. not for category). The advantage of this approach is that only half the features are used for training, testing and applying the model with less computation times.

### 5.4.3 Model training

For the splitting of the data in training- and test-set, the second described method from § 4.4.1 was chosen. This guarantees that in both sets all different kinds of changes can be found and a reasonable training and testing can be executed. The ratio between training- and test-set is 80:20, with 129.142 entries in training and 32.285 entries in test.

The tuning of the hyper-parameters was initialized with the parameters displayed in table 5.3.

| Parameter        | Initial value   |
|------------------|-----------------|
| Objective        | binary:logistic |
| eval_metric      | aucpr           |
| learning_rate    | 0.3             |
| max_depth        | 6               |
| min_child_weight | 1               |
| gamma            | 0               |
| colsample_bytree | 1               |
| subsample        | 1               |
| scale_pos_weight | 10.78           |
| n_estimators     | 1000            |

Table 5.3: Overview of tuning parameters with initial values

The initial parameters are based on the default parameters of XGBoost. These parameters are designed by the creators of XGBoost to deliver good results for many applications. They are a solid foundation to start the tuning. The only differences are the objective, `scale_pos_weight` and the `eval_metric`. Instead of using the default metric "error" (wrong cases/all cases with a threshold of 0.5), it was decided to use a threshold-independent metric like `AUC` or `pr_auc`. This metric is heavily influenced by the expected high number of `TN`. Usually `AUC` is not used as a metric, as

the expected value will be really high in the beginning already. `aucpr`, the area under curve for precision/recall is only focussing on the number of correctly classified positive entries and is used instead.

The tuning process itself was done accordingly to the description of Jain [2016]. Multiple rounds of tuning were performed with changing the parameters separately per round.

The parameters were tuned in the following order with the intermediate results in brackets. The number of estimators was tuned multiple times, as the correct number also depends on the other hyper parameters.

1. `n_estimators` (660)
2. `max_depth` (7) and `min_child_weight` (3)
3. `gamma` (0.0)
4. `n_estimators` (394)
5. `subsample` (0.9) and `colsample_bytree` (0.8)
6. `reg_alpha` ( $1e-5$ )
7. `n_estimators` (394)

The tuning process was done with k-cross-folding, with k being five. In this way more consistent results with a smaller variance are guaranteed. To prevent overfitting, `early_stopping_rounds` was set to 10.

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.

#### 5.4.4 Model evaluation

For evaluations four different thresholds were compared: 0.9, 0.5, 0.1, 0.01. Whereas 0.9 and 0.5 are commonly used as thresholds for binary classifications, the latter were introduced to decrease the number of FN. The calculated best threshold for the model was 0.00094, so that a threshold of 0.001 was additionally tested.

To get more insights, it was decided to apply `predict_proba` on the test-set. The returned probabilities allow a better insight in the data and it is possible to define an own threshold separating change and non-change. It was decided to apply all evaluation methods presented in § 4.5 on the model in order to compare them. Note that the prediction is only working if exactly the input features used for the training are used as an input. However, these columns must not necessarily contain data, they can also be filled with NaN-values.

#### 5.4.5 Locating changes

For the splitting of the polygons the geometric approach was chosen. In contrary to the image segmentation it is independent from the quality of the aerial images and good results with mostly squared polygons are very quickly available. For splitting a QGIS-plugin named `PolygonSplitter`<sup>7</sup> was used. All polygons with an area bigger than  $10.000\text{m}^2$  were split in smaller polygons of roughly equal size of  $1.000\text{m}^2$ . 711 polygons are split in 11.396 new polygons.

Similar to the original polygons, the same change workflow with extracting features was applied to the split polygons. However, no new model was trained, instead the already existing model could be re-used with good results.

<sup>7</sup> for more information see [PolygonSplitter](#).

## 5.5 COMPUTATION TIMES

When describing the computation times it must be differentiated between tasks that need to be executed only once for the creation of the model and the tasks that must be executed again for every change detection. The reported times are derived on an Acer Laptop with a i5 2.3GHz Dual core. For many tasks parallel calculating is possible, so that multiple cores would speed up the complete process.

Two tasks take most of the calculation time, however they must only be executed once:

- Calculating the features

The computation times for calculating the features is correlating mostly with the number of entries. A single feature is calculated very quickly and only makes a small difference. However, the area of a polygon has a big influence. Whereas it took around 50 minutes for the biggest polygon to calculate all features, for the smallest one it took only 20 milliseconds. Based on the average size of a polygon a duration of 1.2 seconds for each entry can be assumed. Having 155.149 polygons, the complete calculation time is around 50 hours, which is also congruent to the experience. Note that this time must be doubled for the entire process, as features must be calculated separately for both years.

- Tuning of the hyper-parameters

The computation time for the tuning depend on three factors. First, the size of the training-set and the number of splits. Second, the number of hyper-parameters with the number of different possible values. Lastly, especially `max_depth` and `n_estimators` can influence the computation times. The tuning described in this chapter took around 20 hours.

All other required tasks for the creation of the model do not have noteworthy computation times and are finished in a range from almost instantly up to maximum 10 minutes.

Getting the results for the test-set when applying the model takes around one minute.

# 6 | RESULTS

In this chapter the results of the change detection algorithm will be presented. Note that only the raw numbers can be found here. A discussion and explanation of the results can be found in § 7.

Figure 6.1 displays how the results could be submitted to a potential customer. He would receive the polygons of the digital map each having a prediction score. The more blue the polygon is, the higher its score. White polygons have a low prediction score. Note that there are many gaps in the data, as only the polygons of the test-set are displayed.

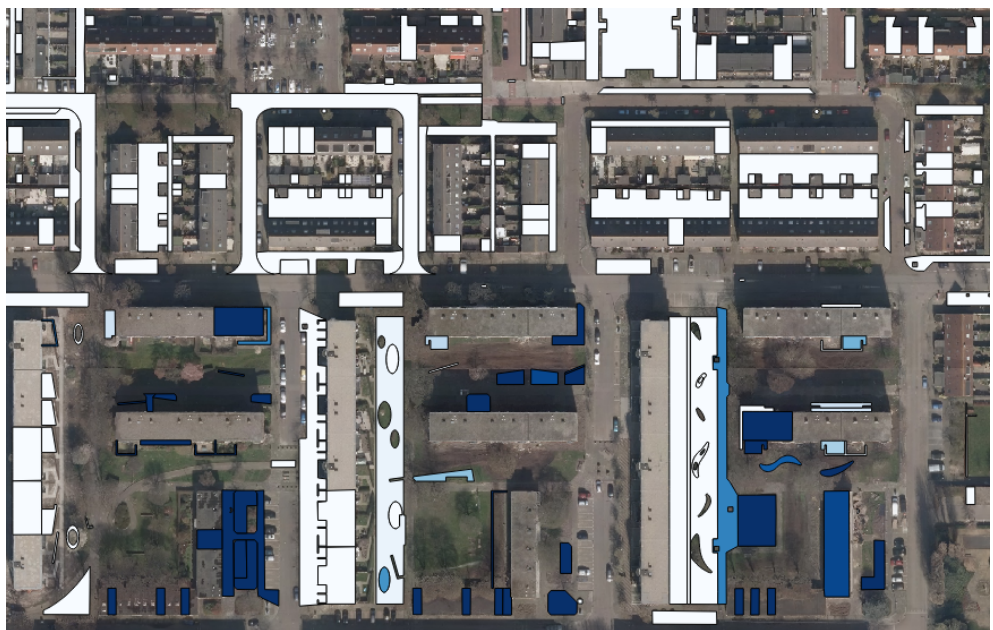


Figure 6.1: Example result of the change detection

Some more example images for successful change detection can be found in the annex at § 8.3.

§ 6.1 presents the results for the test-set that was created during the training of the hyper-parameters. § 6.4 shows the comparison to direct BGT changes and baseline predictions.

## 6.1 MODEL RESULTS

For the XGBoost classification 80% of the data was used for training and 20% of the data was used for testing. Changes can be found in both sets with the same ratio. All results in this section are based on this test-set. All features from § 5.4.2 are used in the model.

### 6.1.1 Confusion matrices

This section describes the **CM** for the described test-set that includes 20% of all polygons and changes. To visualize the effect of changing the threshold, the results of other matrices with different thresholds can be found in table 6.1.

| Threshold | TN     | FP    | FN  | TP  |
|-----------|--------|-------|-----|-----|
| 0.9       | 32.006 | 3     | 160 | 116 |
| 0.5       | 31.996 | 13    | 128 | 148 |
| 0.1       | 31.931 | 78    | 89  | 187 |
| 0.01      | 31.600 | 409   | 63  | 213 |
| 0.001     | 30.248 | 1.761 | 22  | 254 |

Table 6.1: Results for confusion matrix based on different thresholds

For some object-types, change can be detected more easily than for other. To highlight this, table 6.2 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.

| bgt-group | TN     | FP  | FN | TP |
|-----------|--------|-----|----|----|
| building  | 16.478 | 215 | 5  | 90 |
| street    | 6.932  | 585 | 8  | 80 |
| surface   | 5.846  | 585 | 8  | 83 |
| water     | 992    | 17  | 1  | 1  |

Table 6.2: Results for confusion matrix based on different groups

As described in § 1.3 the objective of this thesis is to support change detection and set focus to areas where many polygons are changing. The matrices in table 6.3 show the results of change detection with each a different processing of **FN**. The entry in the first columns shows the maximum distance a **FN** must be next to a **TP** that it will also be classified as a **TP**. Again the thresholds is set to 0.001.

| position | TN     | FP    | FN | TP  |
|----------|--------|-------|----|-----|
| without  | 30.248 | 1.761 | 22 | 254 |
| adjacent | 30.248 | 1.761 | 16 | 260 |
| 10m      | 30.248 | 1.761 | 15 | 261 |
| 25m      | 30.248 | 1.761 | 13 | 263 |

Table 6.3: Results for confusion matrix based on different adjacency levels

### 6.1.2 Scores

This paragraph shows the scores for the model. As the first four scores depend on the threshold, their values are displayed for different thresholds. The last two scores are independent and therefore are only a single value.

- Accuracy

| Threshold | 0.9    | 0.5    | 0.1    | 0.01   | 0.001  |
|-----------|--------|--------|--------|--------|--------|
| Accuracy  | 0.9950 | 0.9956 | 0.9948 | 0.9854 | 0.9448 |

Table 6.4: Accuracy for different thresholds

- Precision

| Threshold | 0.9    | 0.5    | 0.1    | 0.01   | 0.001  |
|-----------|--------|--------|--------|--------|--------|
| Precision | 0.9748 | 0.9193 | 0.7057 | 0.3424 | 0.1261 |

Table 6.5: Precision for different thresholds

- Recall

| Threshold | 0.9    | 0.5    | 0.1    | 0.01   | 0.001  |
|-----------|--------|--------|--------|--------|--------|
| Recall    | 0.4203 | 0.5362 | 0.6775 | 0.7717 | 0.9203 |

Table 6.6: Recall for different thresholds

- F1-Score

| Threshold | 0.9    | 0.5    | 0.1    | 0.01   | 0.001  |
|-----------|--------|--------|--------|--------|--------|
| F1-Score  | 0.5873 | 0.6773 | 0.6913 | 0.4744 | 0.2217 |

Table 6.7: F1-Score for different thresholds

- ROC-AUC The ROC-AUC-score for the model is 0.9843 and is independent from the threshold.
- PR-AUC The PR-AUC-score for the model is 0.7358 and is independent from the threshold.

Figure 6.2 displays all evaluation scores together.

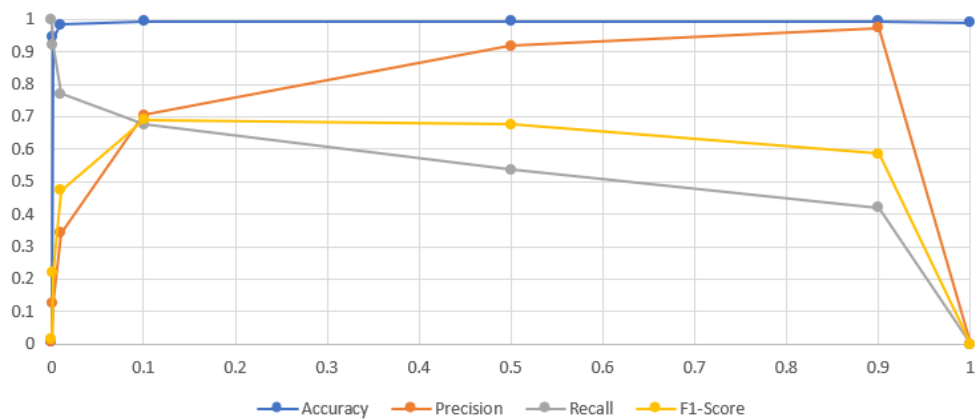


Figure 6.2: Evaluation scores for the model with different thresholds

### 6.1.3 Curves

The following curves display the results of the models independent from a threshold. Three different curves are selected. The first and the last curve look similar, however they are calculated differently and represent different success criteria.

The first curve (figure 6.3) displays the classic ROC-Curve with the inverted FPR (x-axis) and the TPR (y-axis) for different thresholds. The blue line represents the condition where  $TPR = FPR$ .

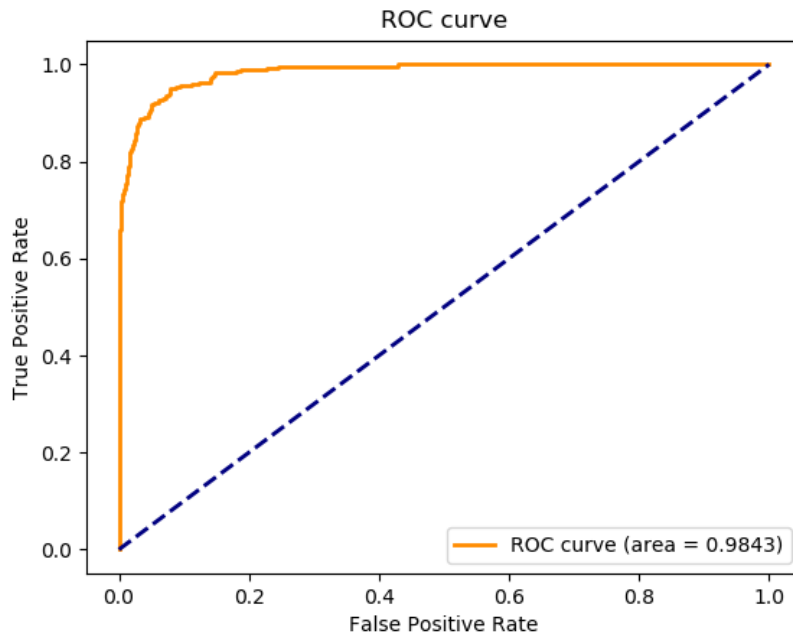


Figure 6.3: ROC-Curve for the model with division (blue line) in good (upper left) and bad (bottom right) classifications

Going right on the y-axis means to lower the threshold and find more FP. Following annotations can be seen for this 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.

The second curve (figure 6.4) 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 constant close to 1, meaning almost no FP are detected but up to 40% of the TP.
- From a threshold of 60% to a threshold to 40% the precision sinks 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% both the numbers of FP and TP are balanced.

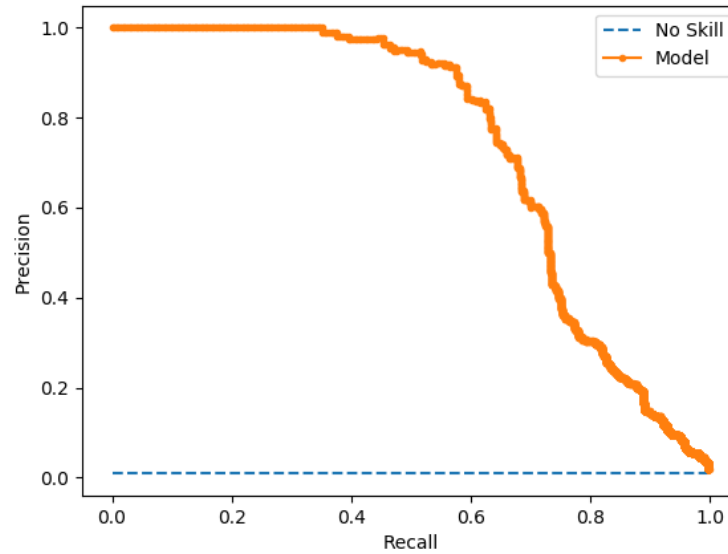


Figure 6.4: Precision-Recall Curve for the model with no-skill curve (blue)

The last curve (figure 6.5) is showing the economic curve of the model. The x-axis is the percentage of all entries and the y-axis is the number of changes. The yellow line represents the model, the blue line marks the point where 95% of the changes are found.

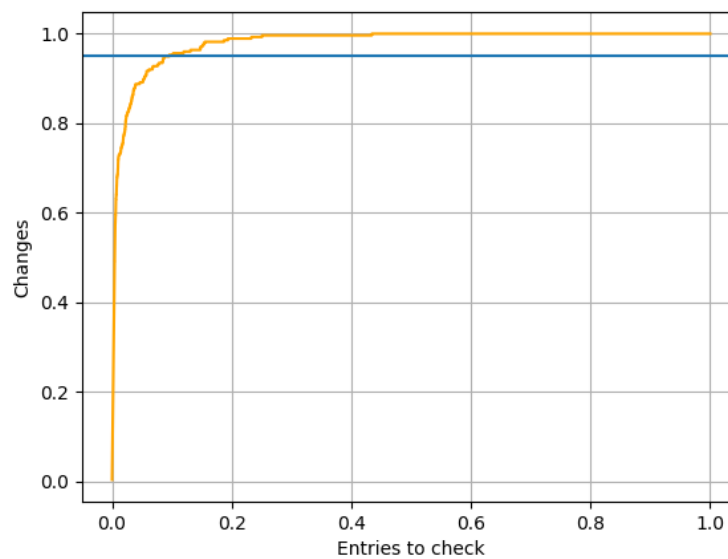


Figure 6.5: Economic curve for the model with 95% bar (blue)

Following annotations can be done for this curve:

- Around 60% of the changes can be found immediately as they have the highest probability scores.
- In order to find 95% (a common contract threshold) of the available changes 10% of all entries need to be checked.
- To find 100% of all changes 25% of the entries must be checked.



## 6.2 FEATURE IMPORTANCE

The feature importance is displayed from both SKlearn and SHAP. However, not all features are displayed, as the corresponding graphs would contain too many entries and are difficult to read. Instead every time the 20 most important features are selected. To compare the relevance for all different models, a model was trained just using the each 20 features of each graph and check their respective PR-AUC-Scores.

### Sklearn

In figure 6.6 the results for the five different importance options are displayed. An enlarged image can be found in the annex. The option weight is the default option that is previewed when using the default feature-importance algorithm from XGBoost.

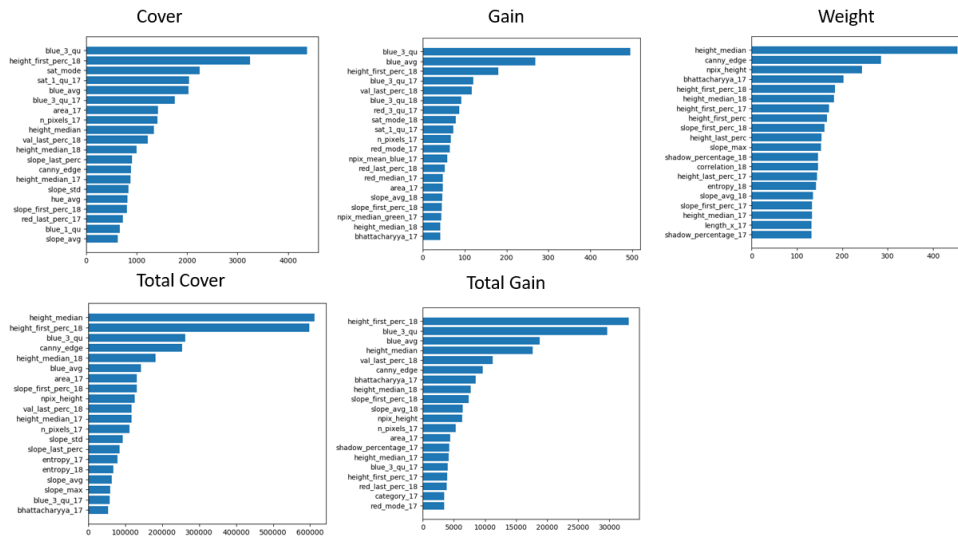


Figure 6.6: Feature importance plots from SKlearn

Figure 6.7 displays the importance scores that were calculated using the SHAP-library.

## SHAP

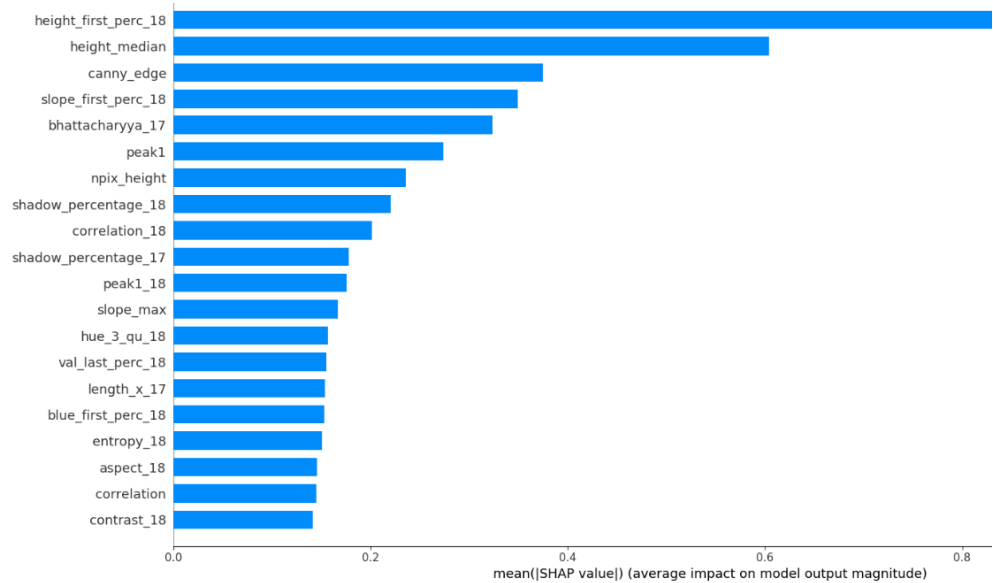


Figure 6.7: Feature importance from SHAP (height\_first\_perc\_18 cut off due to space limitations, original value is 0.95)

### Score

The PR-AUC for different models can be seen in table 6.8. As only the most 20 values are used for training and testing, the scores are lower than for the final model.

| Cover  | Gain   | Weight | Total Cover | Total Gain | SHAP   |
|--------|--------|--------|-------------|------------|--------|
| 0.5459 | 0.6287 | 0.6110 | 0.6275      | 0.6559     | 0.5975 |

Table 6.8: PR-Scores for the different importance options

## 6.3 LOCALIZING CHANGES

The test-set is relatively small and the number of polygons that are bigger than the certain threshold are only a subset of this data. Therefore, evaluations are likely to be inaccurate and need to be repeated with a larger set.

However, some information from the results of the change detection for the split polygons can be extracted. The changes for the split polygons were not available and had to be created by a manual change detection like described in § 4.2.3. The same procedures as for the normal evaluations were applied.

Table 6.9 displays the scores for the split polygons again with a threshold of 0.01 for the first four columns.

| Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|----------|-----------|--------|----------|---------|--------|
| 0.09272  | 0.1913    | 0.7919 | 0.3081   | 0.9351  | 0.5661 |

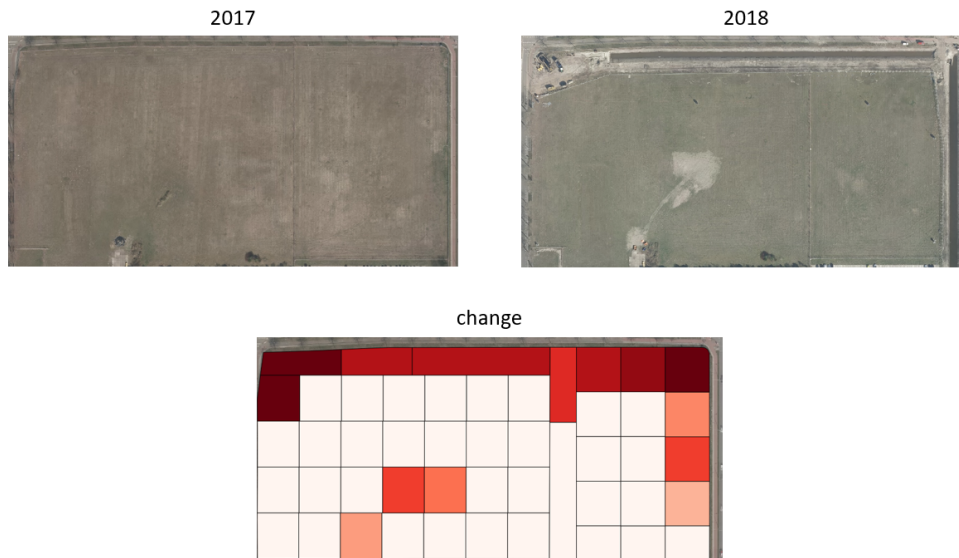
Table 6.9: Evaluation scores for the split polygons

Furthermore, the [CM](#) for the same threshold of 0.01 can be found in [table 6.10](#).

| TN    | FP  | FN | TP  |
|-------|-----|----|-----|
| 9.829 | 740 | 46 | 175 |

**Table 6.10:** Confusion matrix for the split polygons with a threshold of 0.01

[Figure 6.8](#) shows a good example of where the change detection is both helpful in localizing changes and improving the general results (the whole polygon was previously classified as a [FN](#)). The structure of this field is untouched, however in the top and right edge a ditch was dug. The more red a polygon, the higher the probability of a change.



**Figure 6.8:** Example for changes localized with split polygons

## 6.4 COMPARISONS

Even though it is already clear that the algorithm can detect many changes and therefore is able to support the manual change detection process, in this chapter it will be compared to some basic methods to estimate the added value through this method.

### 6.4.1 Comparison to BGT changes

As the digital maps for both years are available, it is possible to extract changes between both files. Whenever polygons from different years are overlapping but classified in a different way, there was a change. The changes directly derived from the map however can only detect changes in which different classes are involved.

However, a comparison between the change detection from the [ML](#) algorithm and the [BGT](#) changes can only give a rough estimate. The algorithm is detecting all polygons that have changed fundamentally, for example in the visual appearance or the height, even if the classes remain the same. The changes directly derived from the map however can only detect changes in which different classes are involved.

There are 5586 overlapping polygons in the [BGT](#) with different categories in 2017 and 2018. However, this high number is caused by inaccurately drawn polygons.

Some polygons were re-drawn in 2018 and do overlap in small parts with adjacent polygons from 2017. A visual check in QGIS also shows that many ostensibly detected changes are no changes in reality.

#### 6.4.2 Comparison to baseline predictions

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 [Brownlee, 2016c].

One of the most basic models that is often used for binary classification is the logistic regression as described by Pant [2019]. A logistic function (an S-shaped curve that predicts probability and with values between 0 and 1, not to be confused with a linear function) is fitted to predict changes of the input data. No logic in learning relations between the features is implied, in its foundation it is just fitting a line to minimize the distance to the residuals. Table 6.11 display the results for this method while using a threshold of 0.4 (maximum F1-score) :

| Score | Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|-------|----------|-----------|--------|----------|---------|--------|
| Value | 0.651    | 0.0198    | 0.8225 | 0.0387   | 0.8199  | 0.0783 |

Table 6.11: Results for logistic regression

Another baseline prediction with more consideration of the small number of changes is the outlier detection model. Changes happen so rarely considering the total number of polygons that they can be considered as outliers. The outlier detection model is specialized in finding outliers in a dataset. Isolationforest is a model from SKlearn and designed for this kind of detection. Table 6.12 display the results for this method while using a threshold of 0.4 (maximum F1-score) :

| Score | Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|-------|----------|-----------|--------|----------|---------|--------|
| Value | 0.8497   | 0.0252    | 0.3007 | 0.0466   | 0.06003 | 0.0146 |

Table 6.12: Results for Outlier detection

It can be seen that in both cases the algorithm developed in this thesis outperforms the baseline predictions across all evaluation criteria.

## 6.5 RECOMMENDATIONS

During the creation and execution of the code occasional obstacles emerged that had to be overcome. These obstacles and their solutions as well as general recommendations for execution are described here, as they can help reproducing the obtained results or when creating an own XGBoost-classification.

- Use seeds  
Whenever data must be shuffled to get a random order or a random subset must be selected from a dataset it is useful to include the seed-parameter. This parameter, usually a number, allows to extract the identical same random order/subset again from a function, provided the seed is the same. For debugging and evaluation the exactly same situation can be restored.
- Copy data  
A basic principle in programming but often disregarded is the usage of copy-statements in python. If the data is copied with an equal sign, only a new reference to the original data is made. Changing the data of this supposedly

copied data will also alter the original data. This can lead to errors in the code. Only a copy statement will really copy the data.

- Use percentiles

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. Figure 6.9 depicts an example, where on a football-field with no visible elevation (left picture) the DSM detects spikes up to 12m (red dots on the right picture). So, instead of using the min/max values for height, the 1% / 99% percentile will be more accurate. Instead of describing the values of spikes, it is then more likely to describe the real height of the data.

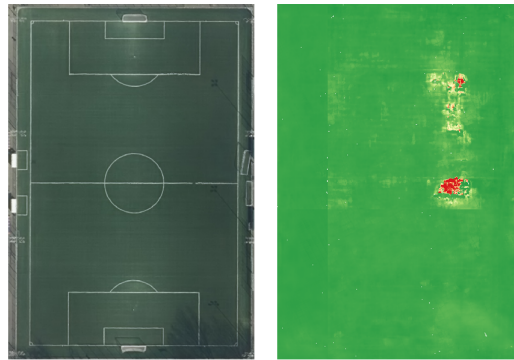


Figure 6.9: Aerial Image from 2018 and the corresponding DSM

- Error handling

Especially for the feature extraction with its long computation time and working with the raw input data the implementation of error handling is necessary. Raw input data can always have errors (for example a polygon with only NaN-values). Without error handling the script is halted. Especially when the algorithm is running on remote servers or during the night this can delay the whole procedure. An algorithm with error handling on the other hand will not crash if an error occurs, but instead save a log entry and continues with the code execution.

- Importance of training

When creating the workflow for a project based on a ML-algorithm, the importance of the training of the algorithm in the methodology should not be underestimated. Even though default parameters will work, adapting these always led to better results. Getting the optimal parameters, that can both deliver good results but are not overfitting is very time consuming, as the model must applied multiple times.

# 7 | DISCUSSION

First of all, it can be concluded that the project is successful and the designed ML algorithm is able to detect changes with the provided input data of aerial images, point clouds and digital maps. Even though it does not find all changes, most changes are labelled correctly with only small number of falsely labeled changes. Considering the total number of polygons, the change detection is suitable to support the manual change detection in a substantial way by directing the focus to certain areas.

In this chapter a final conclusion (§ 7.1) about the method with a particular consideration of the common classification problems (§ 7.2) will be given. The research questions will be answered separately in § 7.3.

## 7.1 FINAL REMARKS

When evaluating the results of the model some general remarks can be made regarding feature importance (§ 7.1.1) and the model results (§ 7.1.2).

### 7.1.1 Feature importance

The first thing that can be noted when looking at the feature importance is that every method for calculating feature importance has a different ranking of important features. It can be explained by the different calculation methods. However, based on own experiences regarding the importance of features and on studies executed by other researchers one ranking method is recommended above all others.

SHAP, even though it has not the highest score in the comparison (see table 6.8), is the both most powerful and reliable tool to calculate the feature importance. It offers multiple methods to evaluate the importance features, let it be bar scales for a global importance of the features or dependency plots to evaluate the feature importance related to another features. Furthermore, in comparison to all other methods it is possible to extract the features' importance just for a single entry with additional information about the direction (higher or lower probability of change) these features are pushing. This information is very valuable when the result for a polygon is unclear. Checking these individual cases can improve the result as a whole.

Even more important is the fact that the SHAP feature importance is the most reliable source for feature information. As described in the papers of [Abu-Rmileh \[2019\]](#) and [Lundberg \[2019\]](#), many other feature importance scores can be misleading. An example for weight would be binary variables. They can consist of valuable information, but have a much smaller number of possible values and therefore are used only once in a decision tree for splitting the values. This is leading to a very low importance for weight. Lundberg is describing the problem regarding the consistency (whenever a different model is applied with more focus on a particular feature, the importance of that feature should not decrease) and accuracy (sum of all feature importances should sum up to the total importance of the model). Only SHAP can satisfy both properties.

However, one should remember that features not selected in the ranking can also be important. The most important features will contribute the most to the results but only together will less important features the change detection can be refined and improved even further. Only in cases in which the feature importance of a feature is zero, it is not contributing to the results. In this case this feature is not considered in the decision tree and removing or keeping this feature will not influence the results.

### 7.1.2 Model results

The current model is able to detect most of the changes in the test-set while keeping the number of FP under an acceptable limit. Regarding the evaluation results following things can be discussed.

Currently the verification of the model is done with a small subset of the same dataset. Even though many measurements are taken against overfitting and the model works properly, a complete clarification if the model is working fine can only be given with testing it on a different dataset. However, in the time frame of this thesis it was not possible to acquire a different research area for verification, as both current and historical data for aerial images as well as maps are required. This combination of data is difficult to obtain for a particular area without customers delivering this data. The final verification will be done as soon as more data is available.

#### Confusion Matrices

When distinguishing between changes and non-changes for a majority of cases the model is absolutely sure if a change has not happened. 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 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 learn better 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 BGT-groups, changes regarding 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 solar panels) the number of FP is lower. That could be explained by the fact that in this group the influence of height differences plays a big role. 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 water ways tends to be the most difficult. However, considering the fact that changes of areas of water are rare, this problem is less eminent.

Looking at the adapted CM it can be seen that around half of the changes are in close relationship to other changes. Usually changes come in clusters, for example a complete quarter is torn off and new buildings are built. The goal of this methods is to set the focus for the people who manually draw the changes. When looking at one change the focus will not only be on the polygons but also in a certain radius around it. It is legit to assume that polygons in a certain radius around another as a change detected polygon will also be seen in a manual detection, so that this adapted matrix can help to show the real potential of this change detection.

#### Scores & Curves

While looking at the scores that depend on the threshold it can be seen that accuracy is really stable and always around 0.99. It is caused by the high number of no

changes that are correctly classified. This evaluation score is therefore less suitable when having such an imbalanced dataset, as false expectations are stoked. This confirms findings of [Brownlee \[2019b\]](#) and [Saito and Rehmsmeier \[2015\]](#). The other scores (Precision, Recall, F1-Score) are all connected to each other, as they are related only to the classification of positive values. At a lower threshold (more **TP**, but also more **FP**) the precision is sinking and the recall is rising. This is expected, as with a lower threshold more **FP** can be found within all positive labelled values (lower precision), but in total more **TP** are found (higher recall). For this project the recall is the more influencing factor, as it is more important to recognize a sufficient proportion of real changes.

The threshold-independent scores and their respective curves can tell something both about the general quality of the model and its results with varying thresholds. Generally, like additionally mentioned in the paper of [Branco et al. \[2016\]](#) and [Saito and Rehmsmeier \[2015\]](#), the **PR-AUC** is more meaningful for imbalanced datasets than the **AUC**. The latter tends to overestimate the quality of the model with similar to accuracy giving very high values due to the high number of negative target values.

The curves show that the model is very sure about classifying of around 50% of the changes, these 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, if considering the small number of recognized changes (true and false) in comparison to the total number of polygons, even the classification with a low precision can be accepted as helpful.

## 7.2 COMMON CLASSIFICATION PROBLEMS

There are many possible reasons why the results of the change detection are not perfect. The most common problems with **FN** are shown in this paragraph. Figure [7.1](#) depicts these images.

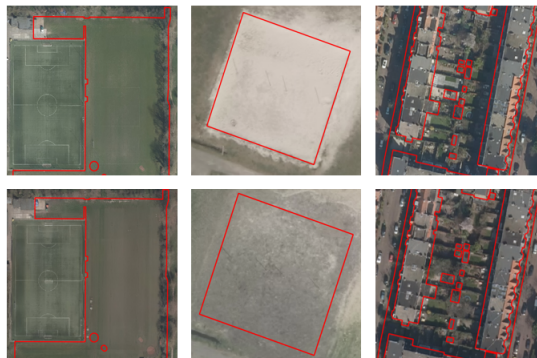


Figure 7.1: Polygons from 2017 (top) and 2018 (bottom) with a positive change label but not being a change

1. Changes too small (left images)  
Sometimes the changes only cover a small part of the complete polygon. The features are mostly influenced by the attributes of the non-changed parts of the polygons and the attributes of the changes have too less influence to identify the change. The change in this polygon is only happening in the top left part.



2. Changes too insignificant (middle images)

For some changes the difference between different years is too small to be recognized by the algorithm. The middle image shows a situation where a patch of sand in 2017 was replaced by concrete with some sandy parts in 2018. This results in similar feature attributes and thus the change cannot be recognized.

3. Polygons with various classes (right images)

Sometimes the polygons from the digital map are not drawn correctly. Even though they should only describe a single type of surface, multiple classes can be found within the polygon borders. The depicted polygons contain gardens as well as small huts and part of the sidewalks. Due to this mixture of classes the feature values have a high standard deviation and information in the features containing changing is harder to find.

## 7.3 RESEARCH QUESTIONS

In this section the research question defined in § 1.2 are answered.

- *To what extent can the change detection be automatized using machine learning algorithms?*

First it can be emphasized that change detection using XGBoost is possible and can be automatized using ML. As described in § 6.1 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 with checking only 5% of all polygons manually.

Under the assumption that the 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 polygons classified as TP. However, the biggest challenges are isolated without a change in height. These are difficult to recognize and cannot be found in 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.

- *Which features of the input data can be used in terms of costs and benefits?*

This question cannot be answered with a simple list of specific features to include and exclude. However, some features seem to have a major influence in every successful change detection as they are appearing in almost every feature importance graph. Furthermore, their calculation does not require additional libraries or progressive algorithms. They are displayed in table 7.1.

Many changes are caused by buildings, which can best be expressed by changes in the height and the slope (a roof usually has a particular angle), which explains the height and slope values of the table. Bhattacharyya is giving an overall impression if a change has occurred and is therefore a good indicator. The features canny edge is given an impression of the structure of a polygon. Edges are caused by different surfaces. The value feature is included, as the value of a colour indicates how much energy is included in this color and allows to distinguish between shadow and no shadow. This value itself will not give estimation for changes, but improves the results for the other colour based parameters.

| Features                   | Number of appearances |
|----------------------------|-----------------------|
| First percentile of height | 6                     |
| First percentile of slope  | 6                     |
| Bhattacharyya              | 5                     |
| Canny Edge                 | 5                     |
| Height median              | 5                     |
| Value last percentile      | 5                     |

**Table 7.1:** Important features and their number of appearances across different importance methods

However, based on personal experience the following features have a high influence succeeding in change detection and are recommended to be used as well:

- Haralick features: Used with success in many image classification algorithms these features were always important.
- Last percentile of height: Similar to the other height-related features it can support the better recognition of building changes.
- Category: The original category influences a lot of what possible changes will look like.

The features mentioned here can all be calculated quickly with no extra efforts needed for extraction. However, category is the easiest feature to get, as it is equivalent to the existing class of the polygon. Bhattacharyya on the other hand needs a separate model for extraction but after the first time initialization no further action is needed.

- *Which information except height can be used from 3D point clouds?*

Next to the standard statistical information from height the following features are used in this model as well:

- Slope
- Aspect
- Homogeneity of height

Using these parameters already 95% of all changes in buildings could be discovered with no need for further investigations in additional features. For this reason and due to time constraints, this question was not further processed. Furthermore, the point cloud used in this thesis is derived from a DSM with the points equally distributed and having no extra information about the surface.

However, when using a point cloud derived from DSM, some features cannot be extracted that could be extracted from LIDAR-point clouds. This includes features like first and last return, point density or intensity. For more information see [Zhu et al. \[2011\]](#).

- *Which metric can be used to evaluate the results?*

Many different evaluation metrics are available for evaluation of the results with different levels of meaningfulness for different situations. In this case especially metrics that work with highly imbalanced datasets are suitable. However, there is not one evaluation metric that will work every time. Depending on what should be explained to whom different metrics are useful.

Following metrics are considered to deliver the most valuable information regarding change detection:

– PR-Curve & PR-AUC

Both the curve and the belonging **AUC** of this curve are especially important during the development of the model. Training a model while monitoring how the **PR-AUC** is changing delivered the best results. In comparison to the **ROC-AUC** it is not converging too fast to the maximum possible score, so that different parameters are still considered (and are not stopped due to early stopping rounds). The **PR-AUC** is well suited to compare different models, as different models will likely give more diverging results (whereas the normal **AUC** will always give values close to 1 for imbalanced datasets)

The **PR**-curve gives a more differentiated view regarding positive results with ignoring the false results. However, this curve is not very suitable for visualizing the outcome of the model for people outside of statistics/ML, as technical terms like Precision and Recall are used. Furthermore, it is not immediately understandable what this curve is telling.

– Economic curve

The economic curve is less important for training and evaluation of the model as it is highly influenced by the number of negative values. However, it might be valuable in a business context.

– Confusion matrix

In comparison to the other two parameters the **CM** has two disadvantages: its results depend on a threshold and different models cannot be compared easily. However, the main advantage of this metric is its easy accessibility for people with less statistical experience. It is clear how many positive and negative values are available in relation to the true numbers.

Furthermore, it is the foundation of many other parameters that can be derived directly from the **CM**.

- *Is it possible to locate the exact position in which the change has happened?*

It is possible to localize the exact position of a change in a polygon. When dividing the polygons in smaller parts and apply the change detection for every sub-polygon a more differentiated image of the polygons with probabilities of change for every part emerges. Note that these polygons are not related to the exact extent of the change, so a reconstruction of the shape of the change is not possible.

## 7.4 CONTRIBUTIONS

The main contributions of this thesis are the following:

- Implementing a holistic change detection as support  
Instead of identifying all changes manually it is now possible to get the probability of change for each polygon. The updating process for digital maps like the **BGT** could be faster and more reliable as unidentified changes are monitored as well. Furthermore, this change detection has a holistic approach, that means it can recognize changes for different classes and is not limited to a certain type.
- Using XGBoost for change detection  
Many papers are focussing on deep learning for change detection, XGBoost is used less for aerial images. This study could prove that gradient boosting

techniques can also be used for change detection and give an indication of the efficiency of these methods.

- **Dealing with temporary changes**

The algorithm is able to deal with changes that are not permanent but could possibly be detected as changes, like cars or seasonal effects. They cause only such a small increase in the probability, so that these polygons are not classified as changes.

## 7.5 FUTURE WORK

This thesis could prove that ML, in particular XGBoost, can be a valuable support for change detection. However, for future work there is still room for improvements.

### 7.5.1 More training data

The most important thing for ML algorithms is the data. 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 happening can be considered. An increased number of positive results (changes) would be especially helpful, as most datasets are very imbalanced with only a fraction of changes. There are two options to get more training data:

#### **New areas**

A simple option is to increase the number of areas used for training. However, new input data must be available and is sometimes expensive. Furthermore, new data also implies new manual change detection and new extraction of features and therefore additional work.

#### **Synthetic examples**

Another option to increase the number of positive results is to create synthetic examples. The easiest option is to just duplicate the positive results, however no new information is added. With SMOTE, Synthetic Minority Oversampling Technique, new examples can be created derived from already existing examples.

### 7.5.2 Deep learning

Deep learning is of increasing importance in the field of ML. For these methods artificial neural networks (ANN) with many hidden layers are used. Input values are processed to get a desired output. How this output is generated is unknown, as the computer selects the important features and algorithms itself. Interest readers are referred to [de Jong and Bosman \[2019\]](#) in which change detection is done using neural networks.

#### **Feature selection**

Deep learning could be used to get many new features as an additional input for this XGBoost-algorithm. A good example for this method is ResNet. It is an ANN trained on special datasets containing thousands of classified images. It takes images as an input and returns a feature vector. Even though these images are trained on photographs instead of aerial images, the created vectors could deliver valuable information and improve the results of the XGBoost based change detection.

#### **Deep learning approach**

Another approach would be to replace the complete XGBoost change detection with deep learning. The three different sources (map, aerial image and point cloud) are given as input data in its entirety instead of derived features. The network should then be able to learn itself which attributes of the input data are important and learn

to classify changes. However, in comparison to XGBoost this approach needs strong GPU-based machines, a lot more processing/debugging and it would be a complete black box to the users (in the sense that neither the features or their importance is known).

### 7.5.3 Creation of a public test-set

The evaluation of the model is a crucial part of the development but also one of the most difficult parts to complete correctly. Not only must a test-set be chosen that has both the right size (not too small or too large) and is representative for the complete data. Furthermore, the evaluation is only valid for this data and a comparison with other methods is difficult. The solution could be a publicly available test-set with example input data and labeled target-data. Every method can be trained with its own data, but for testing, this public set is used. This would allow both an easy-to-do correct testing and comparisons between different methods. These test-sets are already available for many other tasks such as face recognition <sup>1</sup>.

## 7.6 REFLECTION

This section relates the project to the main areas of Geomatics. According to ISO/TC 211, Geomatics is defined as a discipline concerned with (1) collection, (2) distribution, (3) storage, (4) analysis, (5) processing and (6) presentation of geographical data. Except for distribution, all these steps can be found within this thesis:

1. Raw input data was collected by external sources, the collection of features for XGBoost is described in § 4.3.
2. Distribution is not a goal of this project and therefore not a part of this thesis.
3. Data storage and management is mainly done in PostgreSQL as described in § 5.3.1.
4. Analysis of the data is identical with the evaluations of the model as described in § 6.
5. Processing of the data can be seen as the tuning and training of the model as described in § 4.4.
6. Presentation of the data is depicted partially in this thesis with converting the results into polygons visible in QGIS (as for example seen in figure 6.8).

---

<sup>1</sup> see [here](#).

## BIBLIOGRAPHY

- Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (2017). De BGT gebruiken - Basisregistraties - Geobasisregistraties. Available at <https://www.geobasisregistraties.nl/basisregistraties/grootschalige-topografie/de-bgt-gebruiken>.
- Abdullah, A. Y. M., Masrur, A., Adnan, M. S. G., Baky, M. A. A., Hassan, Q. K., and 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.
- Abu-Rmileh, A. (2019). Be careful when interpreting your features importance in XGBoost! Available at <https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7>.
- AlphaBeta (2017). The economic impact of geospatial services.
- Amazon (2020). Model Fit: Underfitting vs. Overfitting - Amazon Machine Learning. Available at <https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>.
- Ban, Y. and Yousif, O. (2016). Change Detection Techniques: A Review. In Ban, Y., editor, *Multitemporal Remote Sensing*, volume 20, pages 19–43. Springer International Publishing, Cham.
- Benedek, C. and Sziranyi, T. (2009). Change Detection in Optical Aerial Images by a Multilayer Conditional Mixed Markov Model. *IEEE Transactions on Geoscience and Remote Sensing*, 47(10):3416–3430.
- Bhattacharjee, J. (2017). Some Key Machine Learning Definitions. Available at <https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48>.
- Bhattacharya, S., Braun, C., and Leopold, U. (2019). A Novel 2.5D Shadow Calculation Algorithm for Urban Environment. In *Proceedings of the 5th International Conference on Geographical Information Systems Theory, Applications and Management*, pages 274–281, Heraklion, Crete, Greece. SCITEPRESS - Science and Technology Publications.
- Bouchaffra, D., Cheriet, M., Jodoin, P.-M., and Beck, D. (2015). Machine learning and pattern recognition models in change detection. *Pattern Recognition*, 48(3):613–615.
- Branco, P., Torgo, L., and Ribeiro, R. P. (2016). A Survey of Predictive Modeling on Imbalanced Domains. *ACM Computing Surveys*, 49(2):1–50.
- Brownlee, J. (2016a). A Gentle Introduction to XGBoost for Applied Machine Learning. Available at <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning>.
- Brownlee, J. (2016b). How to Develop Your First XGBoost Model in Python with scikit-learn. Available at <https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/>.
- Brownlee, J. (2016c). How To Implement Baseline Machine Learning Algorithms From Scratch With Python. Available at <https://machinelearningmastery.com/implement-baseline-machine-learning-algorithms-scratch-python>.

- 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>.
- Brownlee, J. (2019a). Avoid Overfitting By Early Stopping With XGBoost In Python. Available at <https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python>.
- Brownlee, J. (2019b). Failure of Classification Accuracy for Imbalanced Class Distributions. Available at <https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/>.
- Brynnolfsson, P., Nilsson, D., Torheim, T., Asklund, T., Karlsson, C. T., Trygg, J., Nyholm, T., and Garpebring, A. (2017). Haralick texture features from apparent diffusion coefficient (ADC) MRI images depend on imaging and pre-processing parameters. *Scientific Reports*, 7(1):4041.
- Cai, Y. (2003). How Many Pixels Do We Need to See Things? In Goos, G., Hartmanis, J., van Leeuwen, J., Sloot, P. M. A., Abramson, D., Bogdanov, A. V., Gorbachev, Y. E., Dongarra, J. J., and Zomaya, A. Y., editors, *Computational Science — ICCS 2003*, volume 2659, pages 1064–1073. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Cao, K., Guo, H., and Zhang, Y. (2019). Comparison of Approaches for Urban Functional Zones Classification Based on Multi-Source Geospatial Data: A Case Study in Yuzhong District, Chongqing, China. *Sustainability*, 11(3):660.
- Chen, G., Hay, G. J., Carvalho, L. M. T., and Wulder, M. A. (2012). Object-based change detection. *International Journal of Remote Sensing*, 33(14):4434–4457.
- Chen, T. and 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*, pages 785–794.
- Cheng, G. and Han, J. (2016). A survey on object detection in optical remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:11–28.
- Choi, E. and 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 Bie, C., Khan, M., Toxopeus, A., Venus, V., and Skidmore, A. (2008). Hypertemporal image analysis for crop mapping and change detection. In *ISPRS 2008 : Proceedings of the XXI congress : Silk road for information from imagery : the International Society for Photogrammetry and Remote Sensing, 3-11 July, Beijing, China. Comm. VII, WG VII/5. Beijing : ISPRS, 2008. pp. 803-812*, pages 803–812. International Society for Photogrammetry and Remote Sensing (ISPRS).
- de Jong, K. L. and Bosman, A. S. (2019). Unsupervised Change Detection in Satellite Images Using Convolutional Neural Networks. *arXiv:1812.05815 [cs]*.
- Du, S., Zhang, Y., Qin, R., Yang, Z., Zou, Z., Tang, Y., and Fan, C. (2016). Building Change Detection Using Old Aerial Images and New LiDAR Data. *Remote Sensing*, 8.
- Dwivedi, D. (2019). Machine Learning For Beginners. Available at <https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab>.
- Ferreira, H. (2018). Confusion matrix and other metrics in machine learning. Available at <https://medium.com/hugo-ferreiras-blog/confusion-matrix-and-other-metrics-in-machine-learning-894688cb1c0a>.

- Georganos, S., Grippa, T., Vanhuyse, S., Lennert, M., Shimoni, M., and Wolff, E. (2018). Very High Resolution Object-Based Land Use–Land Cover Urban Classification Using Extreme Gradient Boosting. *IEEE Geoscience and Remote Sensing Letters*, 15(4):607–611.
- Gunawardana, A. and Shani, G. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *Journal of Machine Learning Research*, page 28.
- Han, P., Ma, C., Li, Q., Leng, P., Bu, S., and Li, K. (2019). Aerial image change detection using dual regions of interest networks. *Neurocomputing*, 349:190–201.
- He, A., He, J., Kim, R., Like, D., and Yan, A. (2017). An ensemble-based approach for classification of high-resolution satellite imagery of the Amazon Basin. In *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, pages 1–4, Cambridge, MA. IEEE.
- Hegazy, I. R. and Kaloop, M. R. (2015). Monitoring urban growth and land use change detection with GIS and remote sensing techniques in Daqahlia governorate Egypt. *International Journal of Sustainable Built Environment*, 4(1):117–124.
- Hu, Y., Stanley, D., and Xin, Y. (2016). True ortho generation of urban area using high resolution aerial photos. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-4.
- Huo, C., Zhang, S., Li, W., Liu, Q., Zhou, Z., and Lu, H. (2020). Urban change detection based on edge line segments and texture.
- Hussain, M., Chen, D., Cheng, A., Wei, H., and Stanley, D. (2013). Change detection from remotely sensed images: From pixel-based to object-based approaches. *ISPRS Journal of Photogrammetry and Remote Sensing*, 80:91–106.
- Jain, A. (2016). Complete Guide to Parameter Tuning in XGBoost. Available at <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>.
- Jang, Kim, Silva, Kim, Kim, and Chung (2019). Case Study: The Effect of Shade on the RGB Images Obtained from Unmanned Aerial Vehicle in Crop Evaluation. *Journal of Agricultural, Life and Environmental Sciences*, 31(3):143–150.
- Ji, S., Shen, Y., Lu, M., and Zhang, Y. (2019). Building Instance Change Detection from Large-Scale Aerial Images using Convolutional Neural Networks and Simulated Samples. *Remote Sensing*, 11(11):1343.
- Knudsen, T. and Olsen, B. P. (2003). Automated Change Detection for Updates of Digital Map Databases. *Photogrammetric Engineering & Remote Sensing*, 69(11):1289–1296.
- Kraus, K. (1997). *Photogrammetry. Vol. 2: Advanced methods and applications*. Dümmler, Bonn, 4. ed edition.
- Laurae (2017). Visiting: Categorical Features and Encoding in Decision Trees. Available at <https://medium.com/data-design/visiting-categorical-features-and-encoding-in-decision-trees-53400fa65931>.
- Lu, D., Mausel, P., Brondizio, E., and Moran, E. (2004). Change detection techniques. *International Journal of Remote Sensing*, 25(12):2365–2401.
- Lundberg, S. (2019). Interpretable Machine Learning with XGBoost. Available at <https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27>.



- Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67.
- Lundberg, S. M. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.
- Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014). The Evolution of Boosting Algorithms - From Machine Learning to Statistical Modelling. *Methods of Information in Medicine*, 53(06):419–427.
- Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (2018). Basisregistratie Grootchalige Topografie - Basisregistraties - Geobasisregistraties. Available at <https://www.geobasisregistraties.nl/basisregistraties/grootchalige-topografie/basisregistratie-grootchalige-topografie>.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York.
- Murakami, H., Nakagawa, K., Hasegawa, H., Shibata, T., and Iwanami, E. (1999). Change detection of buildings using an airborne laser scanner. *ISPRS Journal of Photogrammetry and Remote Sensing*, 54(2-3):148–152.
- Natekin, A. and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7.
- Nebiker, S., Lack, N., and Deuber, M. (2014). Building Change Detection from Historical Aerial Photographs Using Dense Image Matching and Object-Based Image Analysis. *Remote Sensing*, 6(9):8310–8336.
- Ojala, T., Pietikainen, M., and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987.
- Pafka, S. (2020). Benchmark machine learning. Available at <https://github.com/szilard/benchm-ml>.
- Pang, S., Hu, X., Cai, Z., Gong, J., and Zhang, M. (2018). Building Change Detection from Bi-Temporal Dense-Matching Point Clouds and Aerial Images. *Sensors*, 18(4):966.
- Pant, A. (2019). Introduction to Logistic Regression. Available at <https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148>.
- Parsa, A. B., Movahedi, A., Taghipour, H., Derrible, S., and Mohammadian, A. K. (2020). Toward safer highways, application of XGBoost and SHAP for real-time accident detection and feature analysis. *Accident Analysis & Prevention*, 136:105405.
- Peng, D. and Zhang, Y. (2017). Object-based change detection from satellite imagery by segmentation optimization and multi-features fusion. *International Journal of Remote Sensing*, 38(13):3886–3905.
- Pessoa, G. G., Santos, R. C., Carrilho, A. C., Galo, M., and Amorim, A. (2019). Urban scene classification using features extracted from photogrammetric point clouds acquired by UAV. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13:511–518.
- Popa, C.-A. and Cernăzanu-Glăvan, C. (2018). Fourier Transform-Based Image Classification Using Complex-Valued Convolutional Neural Networks. *International Symposium on Neural Networks*, 10878:300–309.


- Qin, R., Tian, J., and Reinartz, P. (2016). 3D change detection – Approaches and applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, 122:41–56.
- Restrepo, M. (2019). Doing XGBoost hyper-parameter tuning the smart way — Part 1 of 2. Available at <https://towardsdatascience.com/doing-xgboost-hyper-parameter-tuning-the-smart-way-part-1-of-2-f6d255a45dde>.
- Rosebrock, A. (2020). Local binary patterns with python & opencv. Available at <https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/>.
- Rowe, N. and Grewe, L. (2001). Change detection for linear features in aerial photographs using edge-finding. *IEEE Transactions on Geoscience and Remote Sensing*, 39(7):1608–1612.
- Ruzgiene, B. (2012). Requirements for aerial photography. *Geodezija ir Kartografija*, pages 75–79.
- Saito, T. and Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, 10(3).
- Scaioni, M., Höfle, B., Baungarten Kersting, A. P., Barazzetti, L., Previtali, M., and Wujanz, D. (2018). Methods from information extraction from LIDAR intensity data and multispectral LIDAR technology. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-3:1503–1510.
- Shi, J., Wang, J., and 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.
- Singh, A. (1989). Review Article Digital change detection techniques using remotely-sensed data. *International Journal of Remote Sensing*, 10(6):989–1003.
- Song, H.-G., Kim, G.-H., and Heo, J. (2005). Road Change Detection Algorithms in Remote Sensing Environment. In *Advances in Intelligent Computing*, volume 3645, pages 821–830. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Soni, D. (2019). Supervised vs. Unsupervised Learning. Available at <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d-ml>.
- Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer-Verlag, London.
- Taubenböck, H., Esch, T., Felbier, A., Wiesner, M., Roth, A., and Dech, S. (2012). Monitoring urbanization in mega cities from space. *Remote Sensing of Environment*, 117:162–176.
- Teo, T.-A. and Shih, T.-Y. (2013). Lidar-based change detection and change-type determination in urban areas. *International Journal of Remote Sensing*, 34(3):968–981.
- TerraDrone (2019). Comparing drone LiDAR and photogrammetry.
- Théau, J. (2008). Change Detection. In Shekhar, S. and Xiong, H., editors, *Encyclopedia of GIS*, pages 77–84. Springer US, Boston, MA.
- Vignesh, Thyagarajan, and 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.
- Vosselman, G. and Maas, H., editors (2010). *Airborne and terrestrial laser scanning*. CRC Press, United Kingdom.

- Yao, K. (2019). Xgboost bias variance trade-off and hyper-parameters tuning. Available at <https://kehuiyao.github.io/2019/03/21/xgboost-tuning-parameters>.
- Zhang, H., Eziz, A., Xiao, J., Tao, S., Wang, S., Tang, Z., Zhu, J., and Fang, J. (2019). High-Resolution Vegetation Mapping Using eXtreme Gradient Boosting Based on Extensive Features. *Remote Sensing*, 11(12):1505.
- Zhou, Q. (2017). Digital Elevation Model and Digital Surface Model. In Richardson, D., Castree, N., Goodchild, M. F., Kobayashi, A., Liu, W., and Marston, R. A., editors, *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, pages 1–17. John Wiley & Sons, Ltd, Oxford, UK.
- Zhou, X., Chen, Z., Zhang, X., and Ai, T. (2018). Change Detection for Building Footprints with Different Levels of Detail Using Combined Shape and Pattern Analysis. *ISPRS International Journal of Geo-Information*, 7(10):406.
- Zhu, L., Shortridge, A., Lusch, D., and Shi, R. (2011). Feature extraction from 3D lidar point clouds using image processing methods.
- Zong, K., Sowmya, A., and Trinder, J. (2013). Machine Learning Based Urban Change Detection by Fusing High Resolution Aerial Images and Lidar Data. In *Geo-Informatics in Resource Management and Sustainable Ecosystem*, volume 398, pages 522–532. Springer Berlin Heidelberg, Berlin, Heidelberg.
- İlsever, M. and Ünsalan, C. (2012). *Two-dimensional change detection methods: remote sensing applications*. SpringerBriefs in computer science. Springer, London.



## 8.1 EXAMPLE FOR FEATURES

Feature values for the depicted polygon (marked with red) are displayed. Note that not all features but an exemplaric selection was made.



| Colour Features      |        | Height Features   |         | Polygon Features |           | Progressive Features |         |
|----------------------|--------|-------------------|---------|------------------|-----------|----------------------|---------|
| Feature              | Value  | Feature           | Value   | Feature          | Value     | Feature              | Value   |
| red_min              | 103    | height_avg        | 29.774  | n_pixels         | 630       | shadow_percentage    | 7.04    |
| red_max              | 154    | height_first_perc | 24.252  | length_x         | 6.4       | haralick_contrast    | 167.902 |
| red_avg              | 122.98 | height_last_perc  | 31.579  | length_y         | 3.5       | haralick_Entropy     | 87.730  |
| red_first_percentile | 106    | slope_avg         | 14.171  | compactness      | 0.702     | Peaks                | 0.7880  |
| red_last_percentile  | 143    | aspect            | 159.691 | n_vertices       | 5         | LBP[1]**             | 0       |
| red_mode             | 124    | npix_height       | 0.825   | Category         | Building* | LBP[4]               | 7       |
| red_std              | 8.63   |                   |         |                  |           |                      |         |

\* will be converted to a number

\*\* the number corresponds to the position at the histogram

## 8.2 RESULTS FOR UNSUCCESSFUL ATTEMPTS

### no RGB

| TN     | FP    | FN | TP  |
|--------|-------|----|-----|
| 28.814 | 3.195 | 22 | 254 |

| Score | Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|-------|----------|-----------|--------|----------|---------|--------|
| Value | 0.9004   | 0.0736    | 0.9203 | 0.1364   | 0.9735  | 0.6891 |

### no RGB and no HSV

| TN     | FP    | FN | TP  |
|--------|-------|----|-----|
| 27.854 | 4.155 | 21 | 255 |

| Score | Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|-------|----------|-----------|--------|----------|---------|--------|
| Value | 0.8707   | 0.0578    | 0.9239 | 0.1088   | 0.9701  | 0.6538 |

### only difference

| TN     | FP    | FN | TP  |
|--------|-------|----|-----|
| 28.469 | 3.540 | 21 | 255 |

| Score | Accuracy | Precision | Recall | F1-Score | ROC-AUC | PR-AUC |
|-------|----------|-----------|--------|----------|---------|--------|
| Value | 0.8897   | 0.0672    | 0.9239 | 0.1253   | 0.9711  | 0.6427 |

## 8.3 EXAMPLES FOR SUCCESSFUL CHANGE DETECTION



Figure 8.1: Examples for successful change detection for polygons (in red) between 2017 (left) and 2018 (right)

## 8.4 RESULTS FOR THE DIFFERENT FEATURE IMPORTANCE OPTIONS

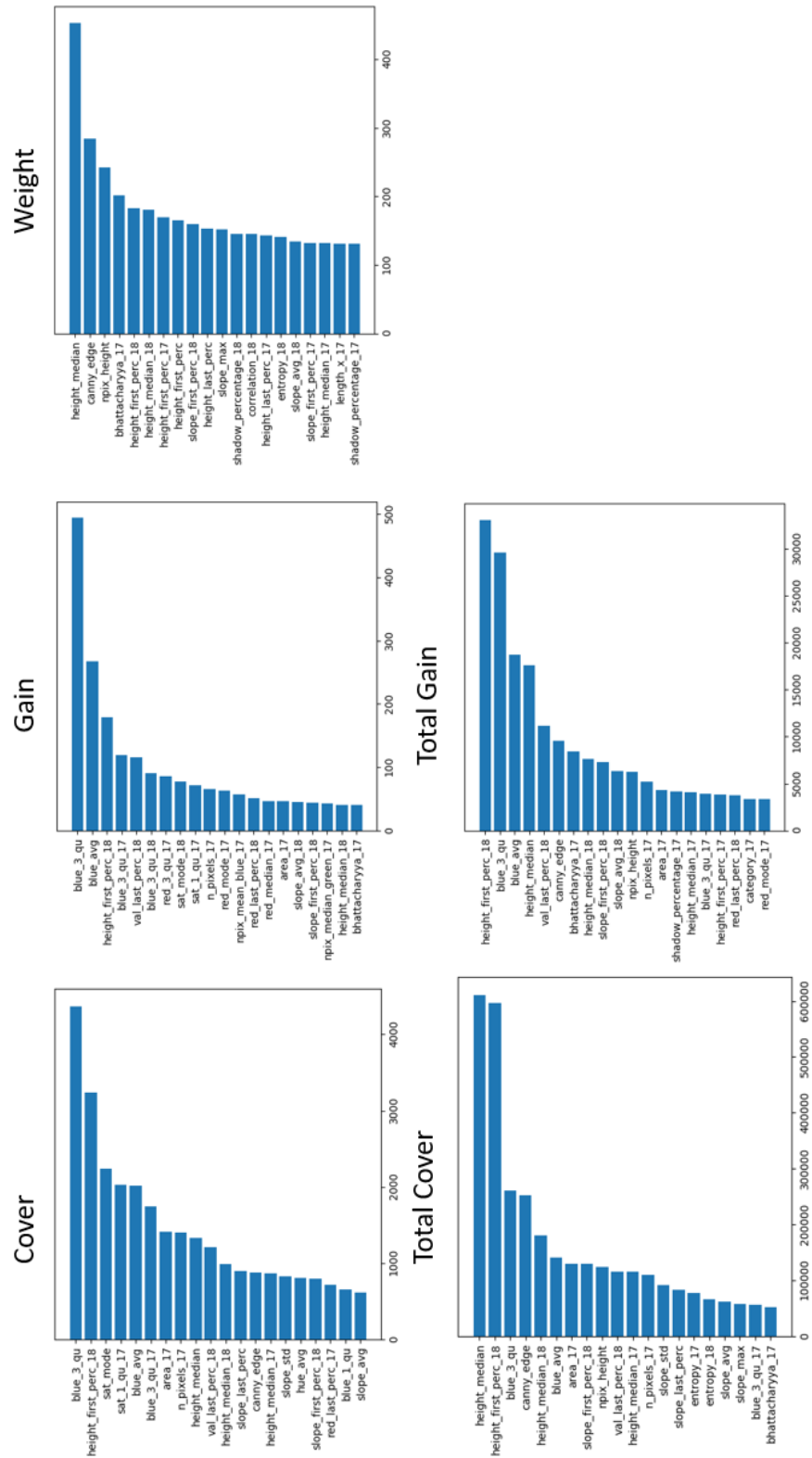


Figure 8.2: Feature importance plots from SKlearn



This chapter provides the exact addresses of the clickable hyperlinks in the thesis as a substitution for a printed version.

- page 1: <https://www.cbs.nl/en-gb/news/2020/05/almost-71-thousand-new-build-homes-in-2019>
- page 1: <https://www.amsterdam.nl/stelselpedia/bgt-index/>
- page 2: <https://readar.com/en/>
- page 5: <https://docs.aws.amazon.com/machine-learning/index.html>
- page 5: <https://www.kaggle.com/kanncaa1/machine-learning-tutorial-for-beginners>
- page 5: <https://statquest.org>
- page 6: <https://www.gemeente.nu/blog/basisregistratie-topografie-werkt-architecten/>
- page 6: <https://readar.com/leads-voor-windmolenfabrikant/>
- page 10: <https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>
- page 16: <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>
- page 19: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0212110>
- page 20: [http://murphylab.web.cmu.edu/publications/boland/boland\\_node26.html](http://murphylab.web.cmu.edu/publications/boland/boland_node26.html)
- page 20: <https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/>
- page 36: [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)
- page 36: [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)
- page 38: [https://en.wikipedia.org/wiki/Hyperparameter\\_optimization](https://en.wikipedia.org/wiki/Hyperparameter_optimization)
- page 42: <https://stackoverflow.com/questions/28719067/roc-curve-and-cut-off-point-python>
- page 46: <https://de.wikipedia.org/wiki/Haarlem#/media/Datei:LocatieHaarlem.png>
- page 47: <https://www.pdok.nl/downloads/-/article/basisregistratie-grootschalige-topografie-bgt->
- page 49: <https://numpy.org/>
- page 49: <https://pandas.pydata.org/>
- page 49: <https://www.scipy.org/>
- page 50: <https://scikit-learn.org/stable/>
- page 51: <https://mahotas.readthedocs.io/en/latest/>
- page 54: <https://plugins.qgis.org/plugins/polygonsplitter/>
- page 73: <http://vis-www.cs.umass.edu/lfw/>

## COLOPHON

This document was typeset using L<sup>A</sup>T<sub>E</sub>X. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.

