FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# ASSIGNMENT OF BACHELOR’S THESIS

| | |
|---|---|
| **Title:** | Satellite image analysis for crop yield prediction |
| **Student:** | Ondrej Pudiš |
| **Supervisor:** | MSc. Juan Pablo Maldonado Lopez, Ph.D. |
| **Study Programme:** | Informatics |
| **Study Branch:** | Knowledge Engineering |
| **Department:** | Department of Applied Mathematics |
| **Validity:** | Until the end of summer semester 2019/20 |

## Instructions

Satellite imagery is a promising tool for crop yield forecasting. In this work we explore different methods and propose new approaches to find correlations between the changes in the images from a given region and its yield production.

1) Study the problem of satellite image analysis.
2) Discuss advantages and drawbacks of different algorithms.
3) Explore various proposed approaches (e.g. deep neural networks) to extract features from satellite images.
4) Compare the performance (classification/training time/scoring time) of the proposed approaches.
5) Consider possible optimizations and improvements.
6) Implement the improved method and analyze achieved results.

## References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 28, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor’s thesis

# Satellite Image Analysis for Crop Yield Prediction

***Ondrej Pudiš***

Department of Applied Mathematics
Supervisor: MSc. Juan Pablo Maldonado Lopez, Ph.D.

May 5, 2019

# Acknowledgements

We express gratitude to the supervisor of this work for his consultancy, interest and willingness during the time of development.

We also thank to the Statistical Office of the Slovak Republic and to the Geodetic and Cartographic Institute of the Slovak Republic for doing a great job in gathering invaluable datasets and providing them to the public.

# Declaration

## Citation of this thesis

Pudiš, Ondrej. *Satellite Image Analysis for Crop Yield Prediction.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019. Also available from: ⟨`https://github.com/ondrejpudis/crop-yield-predictor`⟩.

# Abstract

In this work, we focus on suggesting and implementing a prediction pipeline which allows us to estimate a crop yield. We explore and analyse the Landsat remote sensing collection, extend it with indices that correlate with the vegetation level in order to extract cropland features. We associate these features with the actual crop yield values which are later used for training and testing a regression model. Google's Earth Engine platform plays an essential role in accessing the data and performing complex computations. As the extraction and prediction models, we choose basic machine learning approaches like $k$-means and Linear Regression with the intention of finding out if such models are capable of a good estimation. The result of our work is a tool which predicts crop yields. We test the models on cereals and potatoes datasets. The tests results show that Learning Vector Quantization – Support Vector Machine combination achieves the best results in the cereals dataset with Mean Absolute Error of $0.2836\,\mathrm{t\,ha^{-1}}$ and Learning Vector Quantization with Linear Regression in the potatoes dataset with Mean Absolute Error of $5.3114\,\mathrm{t\,ha^{-1}}$.

**Keywords** remote sensing, machine learning, Google Earth Engine, Landsat, crop yield prediction, cropland feature extraction, Slovakia

# Abstrakt

V tejto práci sa zameriavame na navrhnutie a implementáciu postupnosti krokov, ktorá umožní predikciu úrody plodín. V texte popisujeme a analyzujeme dáta pochádzajúce zo vzdialeného prieskumu Zeme, ktoré rozšírime o indexy vystihujúce vegetačné vlastnosti danej oblasti. Z týchto dát vyberieme podmnožinu úrodných polí. Túto podmnožinu spojíme s reálnymi dátami o úrode a použijeme na natrénovanie a otestovanie regresných modelov. V celom procese má dôležitú úlohu platforma Google Earth Engine, ktorá okrem prístupu k dátam umožňuje aj nad nimi vykonávať rôzne výpočty. V práci volíme základné algoritmy strojového učenia, ako algoritmus $k$-means, či lineárna regresia, so zámerom zistiť, či tieto základné metódy sú schopné dobrej predikcie. Výsledkom našej práce je nástroj, ktorý umožňuje predikciu úrody. Model testujeme na predikcií úrody zemiakov a obilnín. Výsledky testovania ukazujú, že s predikciou obilnín si lepšie poradila kombinácia algoritmov Learning Vector Quantization a Support Vector Machine s absolútnou strednou chybou na úrovni $0.2836\,\mathrm{t\,ha^{-1}}$. Pre úrodu zemiakov nižšiu chybu, $5.3114\,\mathrm{t\,ha^{-1}}$, dosiahol algoritmus Learning Vector Quantization s lineárnou regresiou.

**Klúčové slová** vzdialený prieskum Zeme, strojové učenie, Google Earth Engine, Landsat, predikcia úrody plodín, vyťažovanie charakteristík polí, Slovensko

# Contents

# List of Figures

# List of Tables

# List of Listings

# List of Algorithms

# Introduction

Satellite image analysis has been applied in various use cases throughout the recent decades. It has been helpful in identifying deforestation, monitoring changes in glacier mass, even marking places prone to famines and many others, mostly environmental issues.

Data acquired from the space is a promising source of information on the Earth's properties which can be a game-changer in predicting crop yield or monitoring crop growth. Applying data mining and machine learning techniques on spatial data can lead to prediction models of high quality and accuracy.

The ability to make a reliable prediction is always a considerable competitive advantage. Having a piece of information, even minutes earlier than an opponent can result in earning a large amount of money, avoiding a troublesome or risky situation or preparing for a series of events. This fact means that many people in different roles could benefit from knowing an estimation of yields before a harvest. For example, farmers could approximate their incomes, prepare adequate stocking places, even plan distribution and delivery of their crops.

Those who produce goods could plan to buy the stocks and in case of poor yields, buy the materials before the prices are affected.

The yields predictions can serve a purpose even in the business of reinsurance which is an insurance for insurers. It could help in managing the risks by giving a possibility to weight the chances of catastrophic scenarios carefully.

The main goal of our work is to propose a model that makes a crop yield prediction for the area of the Slovak Republic. Alongside this goal, we explore the data sources, discuss existing approaches, suggest a possible solution, evaluate its results and propose viable improvements of the existing model.

We organise the work in this thesis into ten chapters. It is possible that the theoretical and the practical parts overlay with one another, but it is done to keep thoughts in a purposeful order.

The theoretical part covers a brief introduction into the problem of satellite image analysis and its history. We explain which datasets we use, why we use them and where they come from. We also introduce Google's satellite imagery analysis platform and lastly focus on exploring various algorithms for feature extraction and prediction.

In the practical part, we analyse the available data with the intention of choosing those which contain valuable information. We introduce the prediction pipeline, comment on time consumption enhancement and continue with results analysis and suggestions on improvements.

Finally, we present our software and guidelines on how to install and use it on a reader's own.

# Goals

The main goal of this work is to propose and implement a set of steps (pipeline) which analyses satellite data, extracts the desired information and predicts a crop yield based on it.

In the theoretical part, we target at clarifying the problem of remote sensing and satellite image analysis and defining all the required datasets. We also explore existing methods and propose new approaches to predict crop yields.

In the practical part, we implement the models described in the theoretical part using Google's satellite image analysis tool, compare their performance and prediction errors. Based on the analysis and performance, we propose various improvements and finally present predictions for 2019.

Chapter 1

# Remote sensing

Humankind's need for information about the Earth's characteristics led to the first concepts of remote sensing and continuous Earth observation by a system of satellites.

*Remote sensing* is a process of measuring properties of an object at the Earth's surface remotely. Such measurement relies on the object to reflect or transmit a signal. It can be either acoustical, microwave or optical [1]. The signal is captured by electromagnetic radiation sensors which are capable of extracting data that cannot be seen by a human eye [2]. The result of this process is a set of products called *remote sensing data*.

If placed on a satellite orbiting the Earth it could provide insight on short-term and long-term changes in the environment. Schowengerdt [1] finds the applications for remote sensing data in:

- monitoring the environment,
- defining how the Earth has changed during the last few decades,
- agriculture,
- meteorology,
- mapping,
- exploring renewable and non-renewable energy sources.

Remote sensing has become invaluable in studying biodiversity, deforestation and food security [3].

The idea of remote sensing firstly appeared in the middle of the 19th century, when the first images of the surface were taken from balloons, originally in Paris, later in the USA during the American Civil War. Later on, such images — called *aerial photographs* — were taken from aeroplanes. [2]

*Orbital remote sensing* concepts started to emerge in the middle of the 20th century on several symposiums and conferences concerning this topic, organised by the Environmental Research Institute of Michigan. This initiative was joined by United States Geological Survey (USGS) and later by National Aeronautics and Space Administration (NASA). These joined efforts resulted in launching the first remote sensing dedicated satellite, named Landsat-1[1] in 1972. It carried two sensors — a Multi Spectral Scanner (MSS), designed for spectral analysis and a Return Beam Vidicon (RBV) for cartographic applications. Its principal goal was to capture the remote sensing data, send it to the Earth and offer to users in the simplest way possible. [2]

As a result of human's creative thinking, various analysis of the remote sensing data started to appear. So did the new and better versions of satellites capable of remote sensing. Nowadays, owing to the gigantic technological leap, we are provided with even more precise and frequent data than ever before [2].

NASA itself provides evidence in the form of case studies on how its Landsat data has been used for viewing the speed of glaciers melting, spotting deforestation or locating areas predisposed to flooding [4].

The remote sensing data is available in *collections*. A collection consists of a time series of products. A product can be thought of as an image with pixels containing not only red, green and blue components but also thermal and infrared ones — we call them *bands*. Moreover, there are various metadata on each product indicating cloudiness percentage, location or sensors calibration.

In *an image-centred* analysis, these bands are converted into a regular image and displayed. We only need to map three optional bands onto red, green and blue colours. This can be particularly useful in map creation [1]. Such mapping is shown in Figure 1.1, which maps shortwave infrared, near-infrared and blue bands onto red, green and blue. On the other hand, *a data-centred* analysis focuses on the actual numerical values of the bands, called *features*.

## 1.1 Related research

In this section, we briefly describe four studies related to remote sensing, Google Earth Engine and agriculture.

You et al.'s [5] work offers an innovative approach to the problem of crop yield prediction by applying deep learning techniques such as Convolutional Neural Networks and Long-Short Term Memory. On top of that, they suggest *"a new dimensionality reduction technique"* in order to overcome the sparseness of the training data. This technique is based on an assumption of *permutation invariance* which means that the yield values do not depend

[1]The name was changed by NASA, the original name of this satellite was *ERTS-1*.

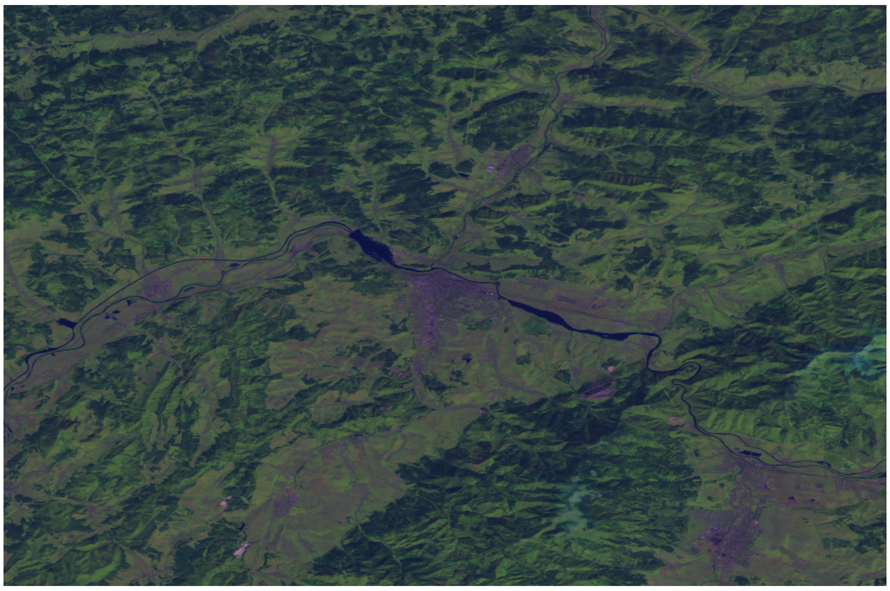


Figure 1.1: An image-centred visualisation

much on the location of the cropland and therefore can be mapped into a histogram of pixel counts. They report that the results achieved by their model outperform the baseline methods like Decision Trees or Ridge Regression by 30%.

Sabini et al. [6] build their work on You's assumptions using the same remote sensing data. They achieve better prediction accuracy than You by using deeper convolutional models. An important finding of their work is that the model can distinguish between soybean and corn croplands. The most informative features are infrared and temperature bands and come from the period between May and October.

A work by Sidhu et al. [7] is concerned with the remote sensing platform Google Earth Engine (GEE). They evaluate platform's performance capabilities concerning spatial aggregations, reductions, raster and vector manipulations of popular remote sensing collections. Their findings prove that GEE is a high-performance platform which processes an average computation in a couple of minutes. What they find challenging is generating a time-series chart over several years (see chapter 3).

Xiong et al. [8] focus on the continent of Africa. They enumerate limitations why this continent remains a challenge for agricultural mapping and address this problem by suggesting *"an automated cropland mapping algorithm (AMCA)"*. This algorithm is capable of automatic croplands identification. The map of the world with croplands identified on it is available at [9]. Creating this algorithm takes three steps:

1. constructing a dataset of high-resolution imagery,

2. building a reference cropland layer,
3. training a Decision Tree algorithm based on the reference from step two.

The reported overall accuracy of this model exceeds 89%.

All the works mentioned above have two common features. First one is the Moderate Resolution Imaging Spectroradiometer (MODIS) remote sensing collection which they use and second is the utilisation of *vegetation indices*. We will use such indices in our work as well (subsection 2.1.1).

Chapter 2

# Datasets

It will appear throughout the work that there are various data which has to be preprocessed, combined and analysed to achieve our goals. In this chapter, we introduce the necessary datasets and collections as well as their sources and institutions which created them.

The most important dataset is a collection of remote sensing data. There are three popular collections:

- Landsat,
- Moderate Resolution Imaging Spectroradiometer (MODIS),
- Sentinel.

Since Sentinel is only a recent initiative of the European Commission and European Space Agency (ESA), it does not provide enough historical data for our research. On the other hand, the quality and resolution of their products are remarkable.

MODIS has been operational since 1999 and it provides the products with either 500 m or 1 km or limited 250 m resolution [10]. Except for the raw products it also covers land surface temperature, snow cover, even thermal abnormalities [11]. Moreover, many researchers, including Sabini [6], You [5] and Huete [10] use this source in their works.

The first Landsat satellite was released to the orbit in 1972 [2]. Since then it has been improved seven times, and now there is the eighth version of Landsat available from 2013. As well as MODIS, Landsat provides both raw and derived collections. They differ only in the resolution. While MODIS offers 250 m to 500 m resolution, Landsat does considerably better with the resolution of 30 m. [11]

## 2.1 Landsat collection

The Landsat collection offers the data in 8 versions, each covers a different period, varies image quality and bands. A listing of all the Landsat versions is available in Appendix D.

The best-suited version for our experiments is *Landsat 7* [12], mainly for these reasons. Firstly, the Landsat 7 products have been available since January 1999 which corresponds with the time availability of crop yield data. Secondly, the bands present in this version can serve as predictors but also can be used for the computation of derived values and indices.

Each data point in the collection represents a squared area huge $900\,\mathrm{m}^2$ and contains the following bands:

- B1, B2, B3: blue, green and red respectively
- B4: near-infrared, it is important for ecology as it is reflected by healthy plants [13]
- B5, B7: shortwaves infrared, they are used for distinguishing between wet and dry areas or various rocks and soils in geology [13]
- B8: panchromatic, a combination of the visible colours into one channel [13]
- B6_VCID_1, B6_VCID_2: thermals infrared, they report the surface temperatures

All the band values are scaled, calibrated and corrected to lower the variances caused by atmospheric anomalies.

As mentioned before, we can use the bands for computation of other values which describe the same area from another point of view. For our work, it makes sense to compute *vegetation indices* which are a transformation of at least two bands. Now, we will have a look at the most popular indices which perform well in estimating the actual vegetation. [10]

### 2.1.1 Normalised Difference Vegetation Index

Normalised Difference Vegetation Index (NDVI) is a stable chlorophyll sensitive index which allows comparison of vegetation changes on both seasonal and annual scale for the local, regional or global environment.

Huete [10] states that NDVI's strength lays in a ratio-based concept of this index which lowers the multiplicative noise. As a critical weakness, Huete considers its non-linearity and influence of various noises, especially atmospheric. Nonetheless, NDVI is still one of the most popular indices to be used in vegetation monitoring [14].

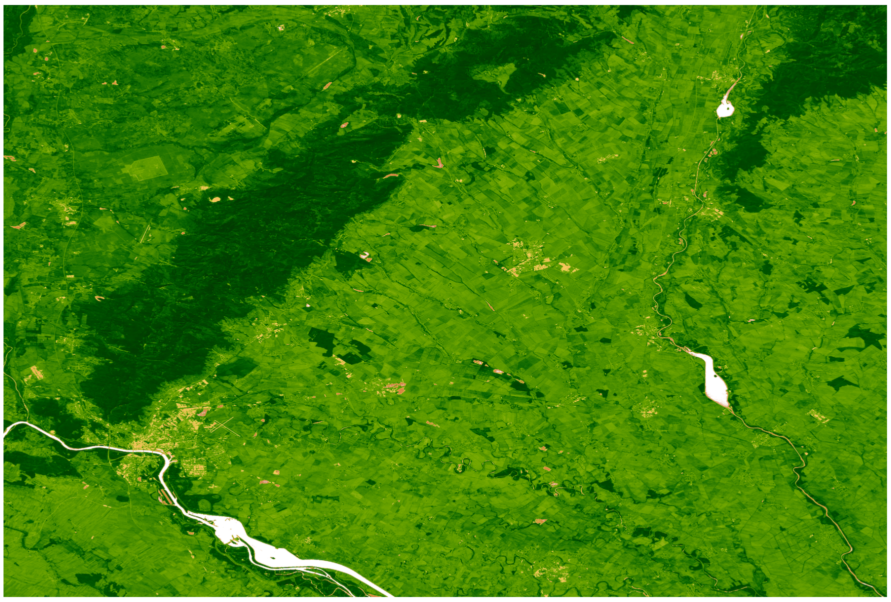


Figure 2.1: NDVI visualisation

The NDVI index is defined by the following formula:

$$\text{NDVI} = \frac{\rho_{near\text{-}infrared} - \rho_{red}}{\rho_{near\text{-}infrared} + \rho_{red}} \tag{2.1}$$

where $\rho_{near\text{-}infrared}$ and $\rho_{red}$ are the surface reflectances of their corresponding Landsat bands [10].

Figure 2.1 visualises the south-western part of the Slovak Republic based on the values of the NDVI index. White areas represent rivers or water reservoirs. Yellow areas are cities and green areas represent vegetation. More accurately, dark green symbolises forests or mountains; light green stands for woods, meadows or croplands. By a closer look, it is possible to see geometric shapes of the croplands.

### 2.1.2 Enhanced Vegetation Index

The Enhanced Vegetation Index (EVI) index was developed in order to improve sensitivity in the regions rich in vegetation and reduce atmospheric impact [10]. Lu [14] concludes that EVI performs poorly in forests and sparsely vegetated areas. He also states that EVI is capable of capturing the inter-annual and seasonal changes.

Figure 2.2 depicts the same location as Figure 2.1. There are only a few differences between these figures. EVI has a problem with distinguishing water areas from city areas. Moreover, the forest and mountain areas are not as dark green as in the NDVI case. These are exactly the troublesome areas mentioned by Lu [14].

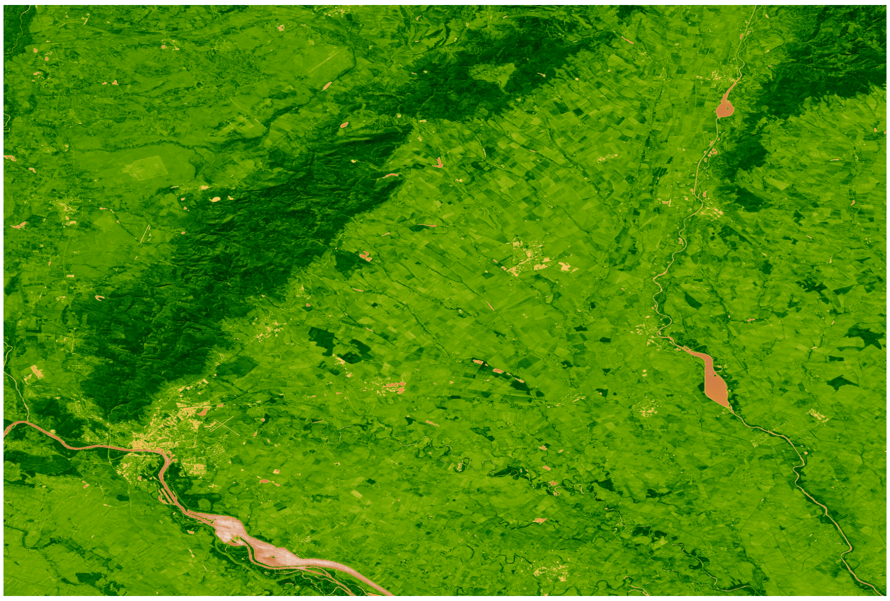


Figure 2.2: EVI visualisation

The EVI equation has the following form:

$$\mathrm{EVI} = G\frac{\rho_{near\text{-}infrared} - \rho_{red}}{\rho_{near\text{-}infrared} + C_1\rho_{red} - C_2\rho_{blue} + L} \tag{2.2}$$

where $G$ is a *gain factor*, $C_1$ and $C_2$ are coefficients of *atmospheric resistance* and $L$ is an adjustment of *the canopy background.* Huete [10], who originally came up with this index, suggests that the values of these coefficients should be 2.5, 6, 7.5 and 1 respectively.

*Atmospheric resistance* is stability against atmospheric aerosol contrasts. The EVI index is one of those which are well resistant to the atmospheric contrasts. [10]

Even though there are many other vegetation indices like *atmospherically resistant vegetation index* or *modified normalised difference vegetation index*, we will use only the two indices mentioned above.

## 2.2 District borders

One of the main goals of our work is to make a prediction for each of the districts of the Slovak Republic separately. Therefore, we need to know where the borders of the districts lay. It is impossible for us to draw the borders on our own. Fortunately, the Geodetic and Cartographic Institute of the Slovak Republic provides a dataset of district administrative borders. [15]

This dataset is available in three formats, one of them is a Geodatabase (GDB). This format is inappropriate for our use case as it contains more than

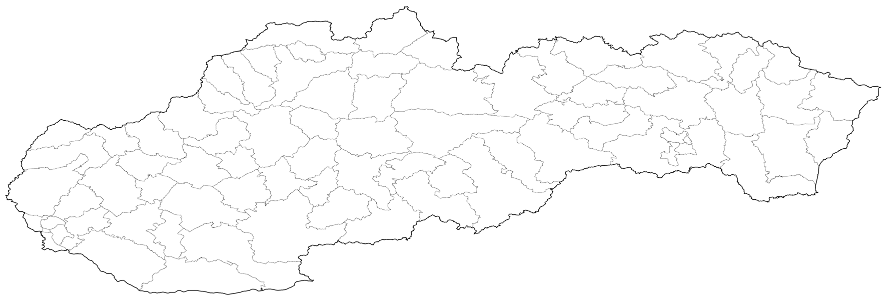


Figure 2.3: District borders

one file. It is much better to have only one, programmatically easily readable file, for example, Comma Separated Values (CSV). Additionally, the Geodetic Institute delivers a conversion service which allows us to convert the GDB source folder into a CSV file.

The resulting CSV file has the following columns:

- **DistrictNumber** a unique identifier of a district
- **DistrictName** a name of the district (e.g. Bratislava I, Martin)
- **RegionNumber** a unique identifier of a region
- **RegionName** a name of the region (e.g. Bratislavský, Žilinský)
- **Area** an area of the district in $m^2$
- **Geometry** a *line ring polygon* of geographic points marking the district border in XML format
- **Shape_Length** a length of the border
- **Shape_Area** an exact area counted from the polygon marked by Geometry

A *line ring polygon* is a round polygon consisting of at least three points which have the same point at the beginning and the end. An example of such polygon is illustrated in Listing 2.1. As the tools which we use in our work do not accept a geometry in this format, we have to convert the original into GeoJSON format. This conversion is performed by a custom utility presented in Appendix E. Listing 2.2 displays the same polygon after conversion.

## 2.3 Past crop yield data

It is never simple to find a dataset with the desired quality and detail. For example, the European Statistical Office (Eurostat) provides many datasets

```xml
<Polygon>
    <outerBoundaryIs>
        <LinearRing>
            <coordinates>
                17.073486,48.174498
                17.073409,48.17453
                17.073739,48.174335
                17.073486,48.174498
            </coordinates>
        </LinearRing>
    </outerBoundaryIs>
</Polygon>
```

Listing 2.1: A geometry in XML format

```json
{
    "type": "Polygon",
    "coordinates": [
        [
            [17.073486, 48.174498],
            [17.073409, 48.17453],
            [17.073739, 48.174335],
            [17.073486, 48.174498]
        ]
    ]
}
```

Listing 2.2: A geometry in GeoJSON format

concerning agriculture, yet none of them contains details on regions of the countries. The same applies to the statistics published by the United Nations (UN).

We found the most adequate and detailed data in the database of the national statistics provider. The Statistical Office of the Slovak Republic publishes the yield data for six crops, namely grains, cereals, oil-plants, potatoes, sugar-beat and fodders. This data currently covers the period from 1997 until 2017 and contains the yield information on national, regional and a district level. The weight unit applied in this dataset is *tons per hectare*, $\mathrm{t\,ha^{-1}}$. [16]

As the Landsat data is not available for years 1997 and 1998, we will ignore these years in the crop yields dataset as well. There is no remote sensing information to make a match.

The data is publicly and freely available online through an application called *DATAcube*. It is a database which contains statistics about business,

| **district** | **1997** | **1998** | **1999** | **2015** | **2016** | **2017** |
|---|---|---|---|---|---|---|
| Slovak Republic | 4.39 | 4.06 | 3.86 | 5.08 | 6.43 | 4.86 |
| Region of Žilina | 3.04 | 3.37 | 3.13 | 4.37 | 4.95 | 4.58 |
| District of Martin | 3.10 | 3.54 | 3.32 | 4.58 | 5.20 | 4.94 |
| District of Ružomberok | 3.21 | 4.03 | 3.66 | 4.24 | 5.55 | 5.20 |
| District of Žilina | 2.31 | 2.63 | 2.39 | 3.97 | 5.14 | 4.15 |

Table 2.1: Cereal yields data in selected years and districts

| **Crop** | $\mathbf{\bar{X}_n}$ | $\mathbf{s_n}$ |
|---|---|---|
| Cereals | 3.6814 | 1.2502 |
| Potatoes | 14.7897 | 7.0433 |

Table 2.2: Selected crops statistics

environment, demography, industry, regions and many others. The *hectare yield of agricultural crops* is available under the *sector statistics*, agriculture, forestry and fisheries tab.

Table 2.1 shows an example of the data in the hectare yields dataset. The data is missing for eight districts in 2011, and seven districts in 2015 which we believe is not much and this dataset can be considered a quality one.

For a future reference and comparison we define the mean value ($\bar{X}_n$) and the standard deviation ($s_n$) of potato and cereal yields in Table 2.2. Later in the text, this will allow us to understand the predicted values and errors better.

There are seventy-nine districts in the Slovak Republic. Nonetheless, the past crop yield data is missing for the districts of Bratislava I – Bratislava V and Košice I – Košice IV. Therefore, we are forced to join the five districts of Bratislava into one for which the crop yield data is available; and similarly the four districts of Košice and the district Košice-surroundings. This operation results in a total number of seventy-one districts used in our work.

## 2.4 Training points

In addition to all the datasets mentioned above, we also need some geographical points to train our models on. For doing so, we use a set of two hundred points, handpicked on the map of Slovakia, keeping in mind diversity of locations, soil types and altitude levels. Afterwards, we assign a variable named `type` to each of these points, indicating what soil type the point is. The options are:

- cropland,
- water,

- city,
- wood.

Each line in this dataset contains a `system index` which is only an internal value of GEE; `land` defines the land type where the point is situated (lowland or highland); `type` indicates a soil type of the given point and `.geo` is a GeoJSON specifying the location of the given point.

```json
{
    "system:index": "1_1_1_1_0",
    "land": "lowland",
    "type": "field",
    ".geo": {
        "type": "Point",
        "coordinates": [
            17.288189725903294,
            48.00745491289555
        ]
    }
}
```

Listing 2.3: A training point

Chapter 3

# Google Earth Engine

It is impossible to carry out the computations required for this work on a single device, even more, if it is a laptop. To work with data as enormous as the remote sensing data undoubtedly is, we need to use an external service that provides enough performance and gives us access to the remote sensing repositories. Google Earth Engine (GEE) is one of those services. In this chapter, we focus on its description, how it works and why we benefit from using it.

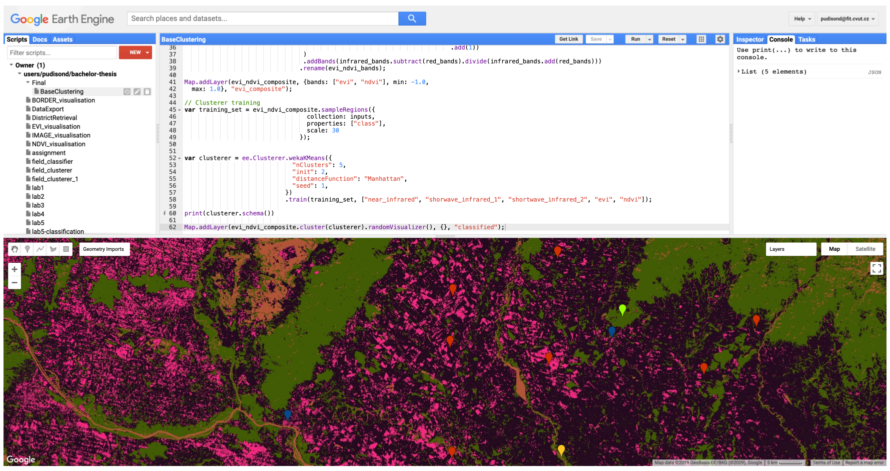
Google itself calls this web-based platform as the most advanced in geospatial processing. It enables large-scale computing and aggregations over an extensive collection of remote sensing data [7].

In [11] they identify the following main components of GEE:

- **datasets** a public catalogue which contains petabytes of remote sensing, weather, topographic, socio-economic and other datasets
- **computational power** a highly paralleled processing
- **APIs** JavaScript and Python interfaces for making requests to GEE servers
- **code editor** a web-browser application which allows fast prototyping and visualisations

GEE makes use of lazy computing which means that a server processes nothing unless a user requested it. The user only composes a chain of functional commands defined by the Earth Engine library which contains more than eight hundred functions. These functions cover simple mathematical operations, geostatistical aggregations, machine learning modelling and image processing. [17]

The lazy computing model introduces confusion to the programming because the GEE objects cannot be used as regular variables in a programme



```
                                                     .add(1))
                              )
                              .addBands(infrared_bands.subtract(red_bands).divide(infrared_bands.add(red_bands)))
                             .rename(evi_ndvi_bands);

Map.addLayer(evi_ndvi_composite, {bands: ["evi", "ndvi"], min: -1.0,
  max: 1.0}, "evi_composite");

// Clusterer training
var training_set = evi_ndvi_composite.sampleRegions({
                                collection: inputs,
                                properties: ["class"],
                                scale: 30
                             });


var clusterer = ee.Clusterer.wekaKMeans({
                            "nClusters": 5,
                            "init": 2,
                            "distanceFunction": "Manhattan",
                            "seed": 1,
                          })
                          .train(training_set, ["near_infrared", "shorwave_infrared_1", "shortwave_infrared_2", "evi", "ndvi"]);

print(clusterer.schema())

Map.addLayer(evi_ndvi_composite.cluster(clusterer).randomVisualizer(), {}, "classified");
```

```
Use print(...) to write to this
console.

▸List (5 elements)                JSON
```

Figure 3.1: GEE code editor

as they only contain a text instruction for the server but not a real value. Accordingly, they cannot be used in loops or conditionals on a local level.

We use this platform because it is capable of performing machine learning operations and is free to use for everybody. Moreover, it provides a public catalogue of the remote sensing data and a Python API. However, there is a drawback in this approach of leaving the computations on a third party — we are restricted to use only those algorithms they have implemented.

For more details and information on how exactly GEE works, we encourage readers to refer to the work of Gorelick and Hancher [17].

## 3.1 Code editor

It is a web-based Integrated Development Environment which allows quick visualisations into both maps and graphs. According to [11], the editor comes with a huge bag of features like:

- code editor which support code-highlighting and syntax corrections for JavaScript
- map for visualising geographic data
- git-based script manager
- drawing tools
- console output, task manager and map inspector

We use this web-based form mainly for testing the ideas, quick prototyping and creating visualisations for this work.

Figure 3.1 illustrates how the web-based editor looks like. In the bottom, there is a map which visualises the code, enables to draw polygons and lines and to pick points. On the left, there is the git-based file manager; the code editor fills the middle while console is on the right.

```
{
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "id": "LANDSAT/LE07/C01/T1"
          },
          "functionName": "ImageCollection.load"
        },
      },
    },
    "filter": {
      "type": "Invocation",
      "arguments": {
        "rightField": "system:time_start",
        "leftValue": {
          "type": "Invocation",
          "arguments": {
            "start": "1999-01-01",
            "end": "1999-12-31"
          },
          "functionName": "DateRange"
        }
      },
      "functionName": "Filter.dateRangeContains"
    }
  },
  "functionName": "Collection.filter"
}
```

Listing 3.1: A payload for GEE

## 3.2 Python API

Nowadays, Python is one of the most popular languages for machine learning. Not only is it easy to use but there are also many libraries available for free which simplify the work of a scientist significantly.

To communicate with the GEE service from a Python code, Google has prepared an API library which is available for download from Python Package Index under the name `google-api-python-client`.

The main purpose of the Python client library is to build the GEE objects, dump these objects into JSON, send the data for computation to the servers and await the results. An example of such JSON is displayed in Listing 3.1. This payload corresponds to the following Python call:

```
ee.ImageCollection("LANDSAT/LE07/C01/T1").filterDate(
    "2017-01-01", "2017-12-31"
)
```

Google does not provide a separate documentation for the Python library. There is only the JavaScript documentation which is supposed to provide the same interface as the Python one.

There is, however, a difference in the way how to evaluate the results. According to the documentation, many objects should implement a method called `evaluate()`, which takes a function as an argument, asynchronously evaluates the object and passes the result to the provided callback function. By evaluation, we mean the computation of the results on a Google server. This approach does not work for the Python client for a simple reason. The server has no means of triggering a local Python function. Therefore, the only way how to evaluate results is a synchronous call to the `getInfo()` method which is available on all objects. Full API documentation is available at [18].

Chapter 4

# Band analysis

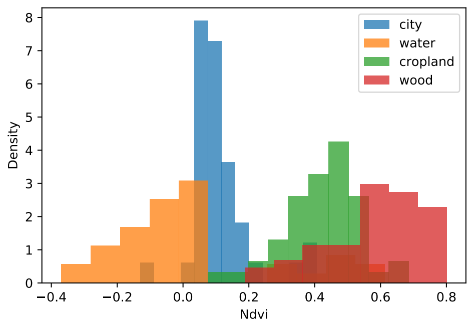
In this chapter, we analyse values of the available bands obtained throughout the time and districts. We also prove that the values of the same areas do not differ much — have a small standard deviation.

As a data source for this analysis we take the handpicked data points which are tagged; therefore we can distinguish between the areas the data points belong to.

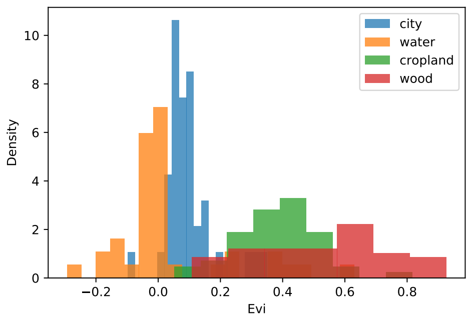
```json
{
    "blue": 34,
    "evi": 0.496424105101188,
    "green": 37,
    "ndvi": 0.46849078,
    "near_infrared": 86,
    "red": 31,
    "shortwave_infrared_1": 31,
    "shortwave_infrared_2": 17,
    "type": "cropland"
}
```

Listing 4.1: An analytic data point

As Listing 4.1 shows all the values except for NDVI and EVI are always integers in range $[0, 255]$ for reflective bands like RGB. Thermal bands would be from another range, but in this work, we do not use any of those. The integer cast is done not only for the presentation and analytic purposes but also for the models to capture the distances better.



(a) NDVI histogram



(b) EVI histogram

Figure 4.1: NDVI and EVI histograms

| **NDVI** | $\bar{\mathbf{X}}_{\mathbf{n}}$ | $\mathbf{s_n}$ |
|---|---|---|
| **city** | 0.1099 | 0.0931 |
| **water** | 0.0252 | 0.2445 |
| **cropland** | 0.4207 | 0.1162 |
| **wood** | 0.5842 | 0.1513 |

(a) NDVI statistics

| **EVI** | $\bar{\mathbf{X}}_{\mathbf{n}}$ | $\mathbf{s_n}$ |
|---|---|---|
| **city** | 0.0955 | 0.0777 |
| **water** | 0.035 | 0.1864 |
| **cropland** | 0.3834 | 0.1377 |
| **wood** | 0.5307 | 0.2181 |

(b) EVI statistics

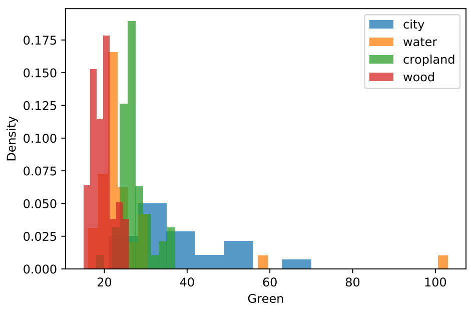
Table 4.1: EVI and NDVI statistics

## 4.1 NDVI and EVI analysis

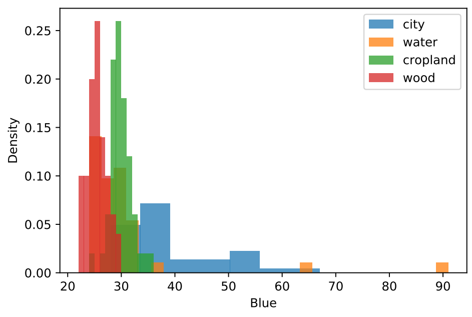
As we mentioned earlier, NDVI and EVI are supposed to be the most promising indices for separating the areas rich in vegetation from the rest. In this section, we examine whether this assumption is valid.

Figure 4.1 shows the histogram of the NDVI and EVI values distribution across all the area types. The vegetation areas (croplands and woods) are well separated from the non-vegetation ones (cities and water areas) by both the indices. On the other hand, distinguishing between woods and croplands is more challenging, especially for EVI as the values overlap much. Determining the water areas is not a problem for both the indices because these values are usually smaller than zero.

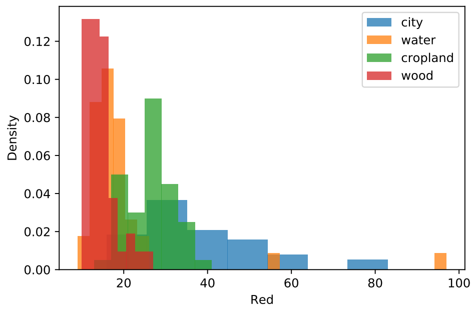
While examining the means and the standard deviations of the indices it appears to us that NDVI performs better in this case. The means are further from each other. Moreover, the standard deviations of NDVIs are smaller which means that the values vary less than they do in case of EVI.



(a) Green histogram



(b) Blue histogram



(c) Red histogram

Figure 4.2: RGB histograms

## 4.2 RGB analysis

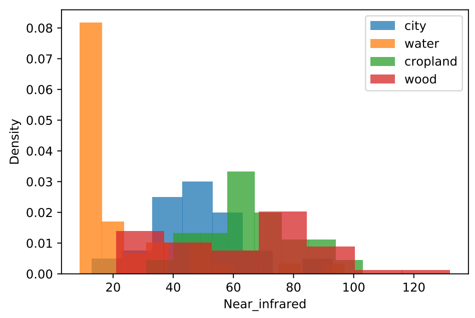
Intuitively it looks like the green component of the spectrum should describe the vegetation very well. Yet Figure 4.2a suggests otherwise. The water areas seem to have the values very similar to both woods and croplands. Not to mention the city areas which get mixed in the range of $[20, 40]$ as well.

From a brief look at the other histograms in figures, 4.2b and 4.2c we see that none of the RGB colours can be used for area separation as it is also a case of these two that the values of all the area types overlay too much and have extensive value ranges. For these reasons we drop the RGB values from the feature extraction process and use them only in the prediction making stage.

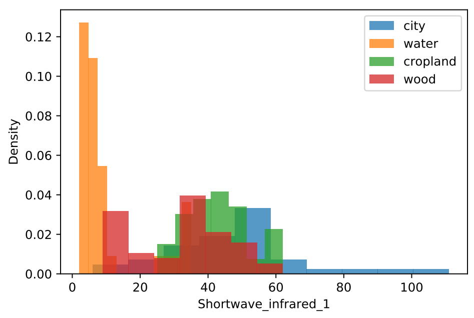
## 4.3 Near and Shortwave Infrared analysis

Near-infrared is a band which is reflected by the healthy plants, more precisely, it is the water in their leaves which scatters these waves back into the sky.

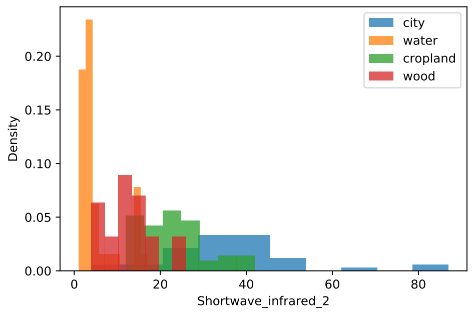
The shortwaves infrared are powerful in distinguishing between wet and dry earth. We can approve this as the water areas obtain small number (the



(a) Near infrared histogram



(b) Shortwave infrared 1 histogram



(c) Shortwave infrared 2 histogram

Figure 4.3: Near and Shortwave infrared histograms

mean is 9.825 with standard deviation 10.2829) while all the other areas obtain values around three times higher (cropland's mean is 42.68).

The water separation applies to the near-infrared histogram as well. However, the wideness of the woods range is quite worrying and overlaps of croplands with cities and woods are not desirable for the feature extraction model. Therefore we drop this band.

## 4.4 Summary

To sum it up, this analysis showed which bands are worth using in the feature extraction and which are not because of their less separative nature. The chosen bands for the feature extraction are *NDVI, EVI, shortwave infrared-1* and *shortwave infrared-2*. This choice does not mean that the rest of the bands will not be used anywhere else. On the contrary, the skipped bands in feature extraction will be used while making predictions.

We also showed that in the chosen bands the values of the same areas do not differ much and are rather situated in a cluster.

Chapter 5

# Feature extraction

Initially, the Landsat collection contains information on both cropland and non-cropland areas. However, we are interested only in that which represents cropland areas. All the other data is irrelevant and can potentially cause a higher prediction error. Therefore, we need to apply an algorithm which filters out the unneeded information. We call this filtering a *feature extraction.*

As there is no tag indicating croplands on the data points the filtering algorithm has to make use of a machine learning approach. Moreover, we assume that the similar areas have similar remote sensing values[2]. So, it is reasonable to apply algorithms which look for similarities between objects and group them up. This approach is called *learning without a tutor* or *unsupervised learning.*

The feature extraction model is trained on the dataset of the handpicked geographical points (section 2.4). Each of the geographical points has these bands available on it: red, blue, green, near-infrared, shortwaves infrared, NDVI and EVI. To optimise the performance of the algorithm we need to choose a subset of these bands to train and categorise on. The band analysis which resulted in establishing this subset is described in chapter 4.

In this chapter, we introduce two algorithms for feature extraction which we will use later in our work.

## 5.1 $k$-means

The $k$-means algorithm belongs to the family of clustering algorithms which is a representative of unsupervised learning methods. The clustering technique separates the given data points into categories so that the similar points are in the same category — *a cluster* [19].

For a given $k$ as the number of clusters and a set of $n$ data points $X \in R^d$, this algorithm separates the data into $k$ clusters $C$.

[2]The evidence for this assumption is provided in chapter 4.

A data point $x_i$ is assigned to the cluster $c_j$ where the distance between $x_i$ and a *centre point (centroid)* of $c_j$ is minimal. To find an optimal solution, we choose the centroids in a way that minimises the squared distance between the points in the same clusters and the respective centroids [20]. This is a so-called potential function $\phi$.

$$\phi = \sum_{x \in X} \min_{c \in C} \|x - c\|^2 \tag{5.1}$$

Finding the exact solution is an NP-hard problem [20]. $k$-means algorithm minimises the value of the potential function iteratively in each step, therefore converges into a local optimum [19] (see Algorithm 5.1).

```
initialise $k$ centroids $C = \{c_1, c_2, \ldots, c_k\}$;
while $C$ changes or stopping condition is not met do
    separate points of $X$ into clusters, so that
      $C_i = \{x \in X \mid i = \arg\min_j \|x - c_j\|\}$;
    find the new centroids $c_i \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} |x|$;
end
```

Algorithm 5.1: $k$-means

To demonstrate the $k$-means algorithm, we present Figure 5.1 which displays the shortwaves infrared data points from the training dataset. These points are neither scaled nor transformed in any way. Figure 5.2 represents the same data points after running $k$-means algorithm on them. Each point has the same colour as all the other points in the same cluster. There is a centroid displayed for each of the clusters.

There appears a question concerning initialisation of the $k$ centroids. Standard practice is to initialise with $k$ random points chosen from a uniform distribution [20]. Since an unfortunate initialisation can lead to poor results, Arthur and Vassilvitskii [20] propose *"a specific way of choosing these centers"* called *k-means++*. Their idea lies in introducing probability to the choice of the next centre. The probability of selecting $x' \in X$ equals to

$$\frac{D(x')^2}{\sum_{x \in X} D(x)^2} \tag{5.2}$$

where $D(x)$ represents the shortest distance from $x$ to the already chosen centroid.

Another effective solution for initialisation, especially of large datasets, is called *canopy algorithm*. It separates the data points into overlapping subsets — *canopies*. A data point is assigned to a canopy if its distance to the centre point is under a specified threshold. This means that one data point can be assigned to more canopies. It is crucial for the canopy distance function to be

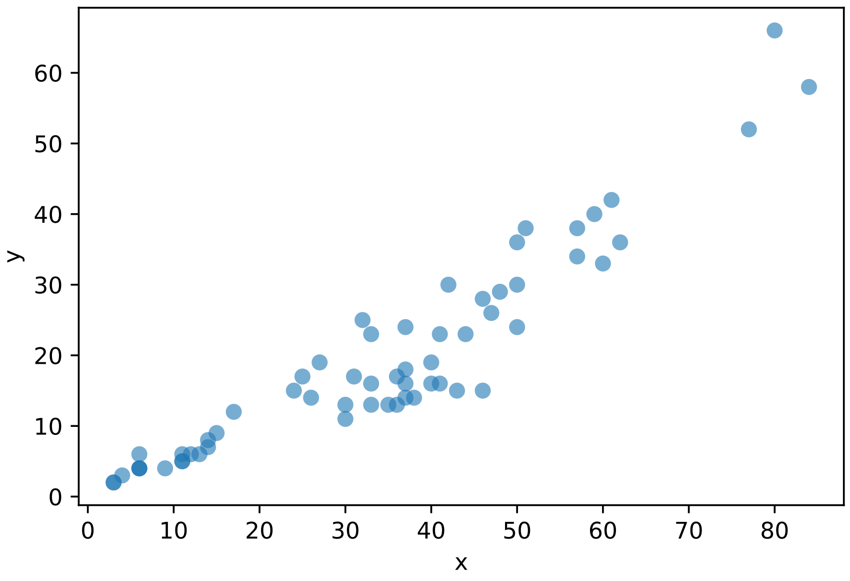


Figure 5.1: $k$-means: data points to cluster

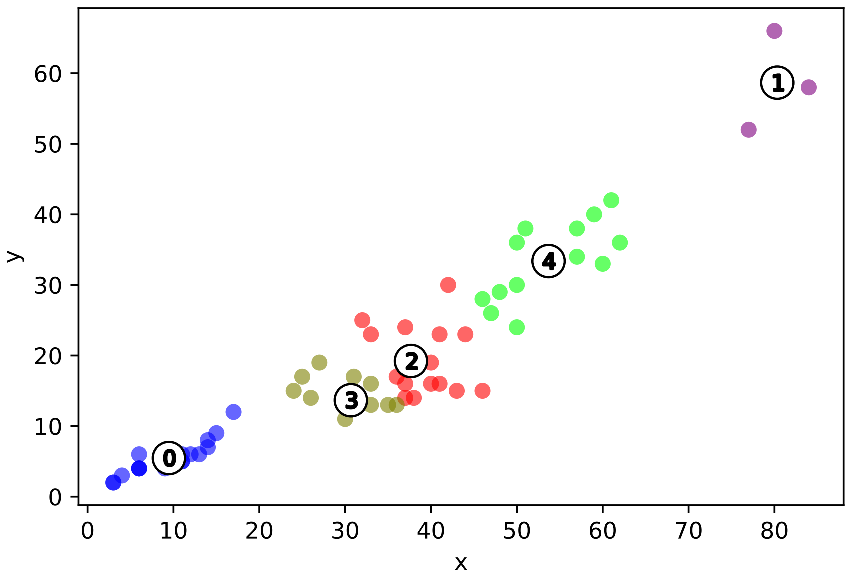


Figure 5.2: $k$-means: the result of clustering

cheap and only approximative. After partitioning into canopies, the algorithm proceeds with standard $k$-means (or any other clustering algorithm), but this time it computes actual distances between those points which appear in the same canopies — they are in the distance small enough to be in a common cluster. [21]

## 5.2 Learning Vector Quantization

Learning Vector Quantization (LVQ) is a representative of competitive learning neural networks. The competitiveness means that each neuron in the network receives the same input, but a response is activated only on the neuron with the highest value of the activation function. In terms of LVQ, a neuron equals to a *codebook vector* $m_i = (\mu_{i1}, \mu_{i2}, \dots, \mu_{in})^T \in R^n$. This approach is often used for detecting patterns in statistical data and approximating borders which separate different groups of data. [22]

Let $k$ be the number of groups (clusters). We assign a set of codebook vectors to each of these $k$ groups. For a vector $x = (\xi_1, \xi_2, \dots, \xi_n) \in R^n$, we assign $x$ to the same group as the codebook vector $m_c$ which has the smallest Euclidean distance from $x$. [22]

$$\|x - m_c\| = \min_i \|x - m_i\| \tag{5.3}$$

By training the model, we understand an iterative process in which we use the training data points to move the codebook vectors either closer or further from the data point [22]. This process can be supervised or unsupervised. We will use the supervised variant to demonstrate the algorithm, although GEE uses the unsupervised one.

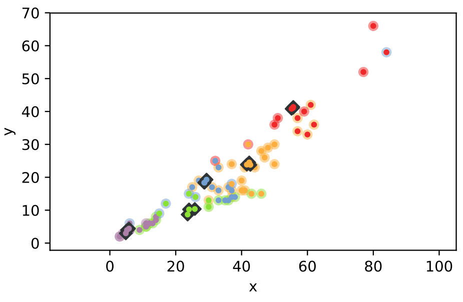
### 5.2.1 Supervised LVQ

Let $t = 1, 2, \dots$ be a series of training epochs. Then the supervised learning process is defined as follows:

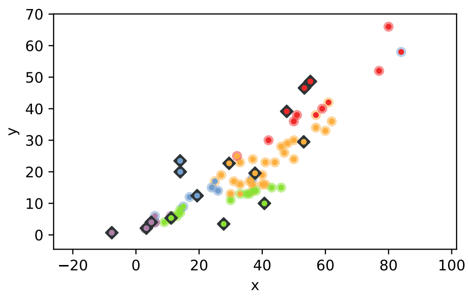
$$\begin{aligned} m_c(t+1) &= m_c(t) + \alpha(t)[x(t) - m_c(t)]; \quad x \in G_c \wedge m_c \in G_c, \\ m_c(t+1) &= m_c(t) - \alpha(t)[x(t) - m_c(t)]; \quad x \notin G_c \vee m_c \notin G_c, \\ m_i(t+1) &= m_i(t); \quad i \neq c \end{aligned} \tag{5.4}$$

where $x$ is a training point, $m_c$ is a codebook vector with the shortest distance to $x$ and $\alpha(t)$ is a learning-rate factor function (usually a number from $[0, 1]$). $G_c$ is a group which both $m_c$ and $x$ belong to. [22]

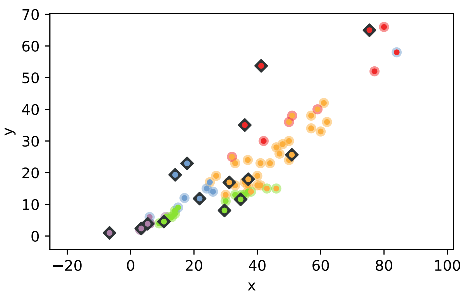
Although this is an optimisation problem of finding the decision borders such that the misclassification rate is minimised, the real implementation is based on an iterative approach of going over the training dataset for $t$ epochs and moving the closest codebook vectors.



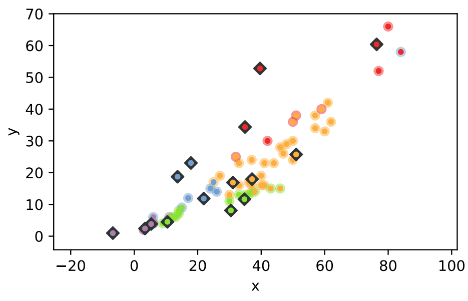
(a) One iteration



(b) Ten iterations



(c) Fifty iterations



(d) A hundred iterations

Figure 5.3: LVQ classification process

Figure 5.3 depicts how the positions of the codebook vectors (black diamonds) change throughout iterations. Each data point has two colours, the inner colour is the predicted label, and outer circle colour is the target one.

### 5.2.2 Unsupervised LVQ

The same iterative process as in supervised variant applies here. The main difference is in the way how codebook vectors are moved. As we do not know which codebook vectors and training points belong together, we only move the closest vector codebook even closer to the training point.

$$m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)] \tag{5.5}$$

Chapter 6

# Prediction algorithms

In this chapter, we define the prediction models which we will use for making the actual predictions from the data we receive from feature extraction. In this case, we will be solving *a regression.* A problem is marked as a regression one if the predicted variable is continuous — it can be assigned a limitless number of values from $R$. A regression problem is a representative of the *supervised learning* algorithms.

This chapter focuses on an introduction into two regression algorithms used in this work.

## 6.1 Linear Regression

Consider $x \in R^n$ a vector of $n$ predictors and $y \in R$ a continuous unknown variable — called *a response* or *a dependant variable.* Linear Regression lets us model a linear dependency of $y$ on $x$ [23]:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \epsilon \tag{6.1}$$

where $\epsilon$ is a normally distributed random variable and $w_0$ is an intercept (a constant move in the direction of the $y$ axis). $w = (w_0, w_1, \ldots, w_n)$ is a vector of *the regression parameters.*

For a Linear Regression prediction, we need to know what is a good estimation of the vector $w$, noted as $\hat{w}$. Then, the prediction has the following form:

$$\hat{y} = \hat{w}^T x \tag{6.2}$$

Finding a good estimation of the values of $w$ is the goal of a training stage. For the best $\hat{w}$ applies that the differences between the correct responses and their predictions are the lowest possible. This intuitive idea is transferred into an optimisation problem of minimising a residual sum of squares (RSS).

$$RSS(\hat{w}) = \sum_{i=0}^{p} (y_i - \hat{y}_i)^2 = \sum_{i=0}^{p} (y_i - \hat{w}^T x)^2 \tag{6.3}$$

where $p$ is the number of training data points, $y_i$ is the real value of the $i$-th data point and $\hat{y}_i$ is a prediction for it. [23]

We are looking for a minimum of a function consisting of $n$ variables. One of the possible approaches is a method called *gradient descent*. It is based on a recurrent equation:

$$\hat{w}^{(i+1)} = \hat{w}^{(i)} - \alpha \nabla RSS(\hat{w}^{(i)}) \tag{6.4}$$

This equation generates a sequence of vectors $\hat{w}^{(i)}$ which converges into either global or local optimum. [23]

$\nabla RSS(a)$ is a *gradient* of the function $RSS$ in the given point $a$. Gradient is a vector of partial derivations in $a$.

$$\nabla RSS(a) = \left( \frac{\partial RSS}{\partial x_1}(a), \ldots, \frac{\partial RSS}{\partial x_n}(a) \right) \tag{6.5}$$

By $\nabla RSS$ we understand a function which assigns a gradient to each point of the function's domain.

To prevent a *colinearity collision* which appears when the columns of $X$ are almost linearly dependent, we often introduce a *regularisation* term into the equation in the form of penalisation [24]. There are two types of regularisation. The first one is an $L_1$ regularisation, also called *Lasso Regression* which adds a sum of coefficients in $\hat{w}$, excluding intercept.

$$RSS_{L_1}(\hat{w}) = \sum_{i=0}^{p} (y_i - \hat{w}^T x)^2 + \alpha \sum_{i=1}^{n} \hat{w}_i \tag{6.6}$$

An $L_2$ regularisation, *Ridge Regression*, sums the coefficients squared.

$$RSS_{L_2}(\hat{w}) = \sum_{i=0}^{p} (y_i - \hat{w}^T x)^2 + \alpha \sum_{i=1}^{n} \hat{w}_i^2 \tag{6.7}$$

So, while finding an optimum, we are looking for a one which has the coefficients as small as possible. The regularisation term is always weighted by a parameter $\alpha$. If $\alpha = 0$, we get the standard linear regression. [24]

## 6.2 Support Vector Regression

The Support Vector algorithms were originally developed for pattern recognition and classification by drawing a decision boundary, called *hyperplane*, between sets of data such that the margins between different sets would be the largest possible. The data points which are closest to the boundary are called *support vectors*. [25]

Support vector algorithms are capable of function estimation as well. The idea is to find a function (not necessarily linear)

$$f(x) = w^T x + b; \quad w, x \in R^n, b \in R \tag{6.8}$$

which has at most $\epsilon$ deviation from the $y_i$ being a training data points and is the flattest possible [26]. It means that we accept an error smaller than $\epsilon$ and do not penalise it. The idea leads to the following loss function:

$$|y_i - f(x_i)|_e = \max\{0, |y_i - f(x_i)| - \epsilon\} \tag{6.9}$$

where $f : R \to R$, $x_i \in R$ and $y_i \in R$ is a target value for $x_i$. It is called an *$\epsilon$-insensitive loss function* and was firstly introduced in the $\epsilon$-SVR algorithm [27]. In this section, we assume only one dimensional space for simplicity.

As it is suggested that choosing a good $\epsilon$ can be a tricky task, in order to avoid it we use an algorithm called $\nu$-SVR which introduces a $\nu \in (0, 1)$ parameter which participates on automatic computation of $\epsilon$, hence controls the number of errors and the number of support vectors, as well. [27]

The $\nu$-SVR algorithm's target is to minimise the following function $\tau$:

$$\tau(w, \xi, \xi^*, \epsilon) = \frac{1}{2}\|w\|^2 + C(\nu\epsilon + \frac{1}{\ell}\sum_{i=1}^{\ell}(\xi_i + \xi_i^*)) \tag{6.10}$$

subject to

$$\begin{aligned} (wx_i + b) - y_i &\leq \epsilon + \xi_i \\ y_i - (wx_i + b) &\leq \epsilon + \xi_i^* \\ \xi_i \geq 0, \xi_i^* &\geq 0, \epsilon \geq 0 \end{aligned} \tag{6.11}$$

where $w, b \in R$ are parameters of a linear function, $\ell$ is the number of training points and $C$ is a regularisation constant. [27]

The presented regression is linear. However, it is possible to generalise this approach and turn the regression into a non-linear one. This transformation is achieved by the so-called *kernel* method which maps training points into some other feature space, $\Phi : X \to F$. [26]

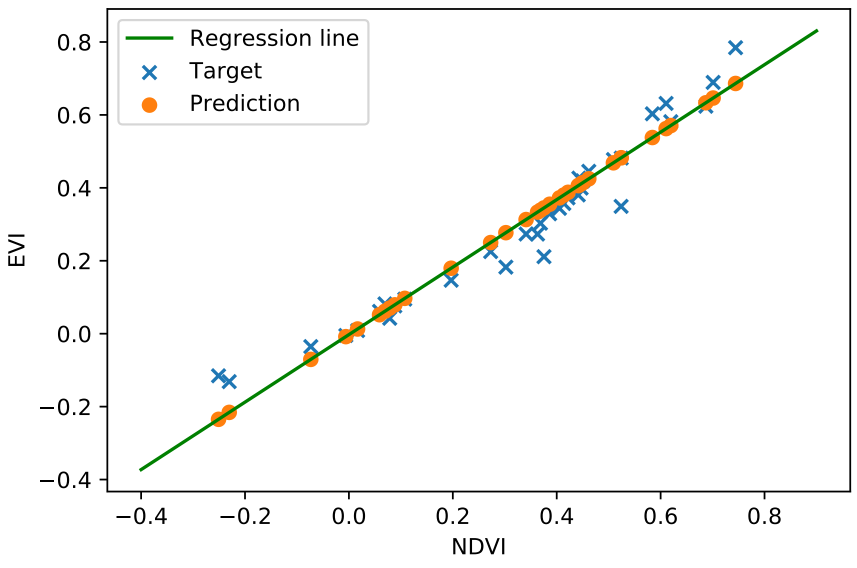
The most frequently used kernel functions are:

- linear: $k(x, y) = xy$
- polynomial: $k(x, y) = (\gamma xy + b)^{degree}$
- Radial Basis Function (RBF): $k(x, y) = \exp\frac{-\|x-y\|^2}{\gamma}$
- hyperbolic tangent: $k(x, y) = \tanh(\gamma xy + b)$

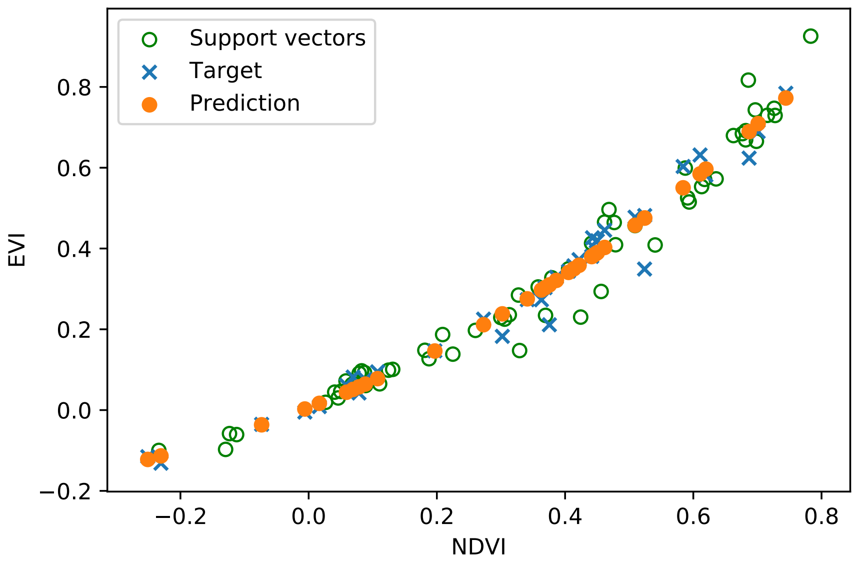
where $\gamma$ along with $b$ are the kernel coefficients.

## 6.3 Models comparison

The plots in this section are only illustrational without connection to the rest of the work. They show how the models work by plotting their predictions of



(a) Linear Regression



(b) $\nu$-Support Vector Regression

Figure 6.1: Models comparison

EVI based on NDVI. The blue crosses represent the correct values while the orange points are the predictions.

Linear Regression (figure 6.1a) is initialised with no parameters. The green line which lays over the prediction points is the *regression line* which has a form of a linear equation $y = ax + b$ where $a$ is the weight from $\hat{w}$ and $b$ is an intercept.

$\nu$-SVR (figure 6.1b) is initialised with a RBF kernel, $\nu = 0.5$, $\gamma = 0.15$ and $C = 100$. Because of the RBF kernel the predictions of $\nu$-SVR do not lay on a straight line but rather on exponential which is caused by the fact that the RBF kernel maps by utilising a $e^x$ function.

As the target points also do not lay on a line, the non-linear $\nu$-SVR predicts these points even better than the Linear Regression model does.

## 6.4 Error measurement

Unlike classification which, in most cases only counts how many elements were classified correctly, in regression modelling we need to keep in mind its continuous nature. For exploring a regression model error, we need to count the difference between predicted and the actual values. There are more ways for doing it, but the most common ones are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) [28].

In this work, we use MAE over RMSE because Willmott et al. [28] suggest that this measure is *"unambiguous and natural measure of average error magnitude"*. Moreover, the RMSE value is hardly interpretable, and the results tend to become increasingly higher than MAE. Willmott et al. [28] also state that RMSE should not be used in a model comparison because there is no consistent relationship between RMSE and average model error. Additionally, MAE allows us to understand the results better and relate them with a dataset's mean and standard deviation.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} \tag{6.12}$$

where $y_i$ is a target value and $\hat{y}_i$ is a prediction for it.

Chapter 7

# The model

Now, that all the prerequisites are defined, we can present the prediction pipeline itself. Firstly, we give a quick overview and then focus on feature extraction, an association of the features with past crop yields and finally making a prediction.

Let `years` be a given range of years. This range must not exceed the 1999 to 2017 boundary. For each of the years, we build a separate feature extraction model which is provided with a collection of Landsat features and a subset of the handpicked geographical points used for training. After being trained, the feature extraction model returns a collection of features which represents croplands in the given district. Except for the regular bands, the Landsat feature collection is also extended by EVI and NDVI values.

Now, assume `districts` being a list of all districts in the Slovak Republic. For each of them, we create a new prediction model, train it and make some predictions. Then, we compute MAE — an estimated model error. For this estimation to be unbiased, training data must not be the same as testing data. Therefore, before training the very first model, we split `years` into `training_years` and `testing_years`.

To present the idea of our prediction pipeline graphically, we created a diagram which is available in Appendix C.

## 7.1 Image collection

To perform an analysis, we need to acquire the data firstly. We access the Landsat collection `ee.ImageCollection("LANDSAT/LE07/C01/T1")` and filter it to contain only images for a requested year. To proceed we compose the whole `ee.ImageCollection` to a single `ee.Image`.

```
ee.Algorithms.Landsat.simpleComposite(collection=self.collection)
```

This algorithm applies calibrations, selects the scenes with the lowest possible cloud score and aggregates the accepted pixels.

Now, the image contains all the bands defined in section 2.1. The last step before passing the `ee.Image` to the feature extraction we calculate the vegetation indices and add them to the image.

## 7.2 Feature extraction

For feature extraction, we use clustering algorithms which are located in the GEE package `ee.Clusterer`. The offer of algorithms consists of:

- standard $k$-means (`wekaKMeans`),
- Learning Vector Quantization (`wekaLVQ`),
- Cobweb clustering algorithm (`wekaCobweb`),

and other derivatives from these algorithms which either take the exact number of clusters to form or are capable of finding the best number of clusters themselves during the training phase. In our work, we use the two models described in chapter 5.

$k$-means algorithm instance is initialised with the number of clusters, centroid initialisation method and distance function. The options for the $k$ centroids initialisation are:

- random
- k-means++
- canopy

As a distance function we can choose:

- Euclidean distance $d(x, y) = \sqrt{\sum_{i=0}^{p-1} (x_i - y_i)^2}$,
- Manhattan distance $d(x, y) = \sum_{i=0}^{p-1} |x_i - y_i|$,

where $x, y \in R^p$. In our implementation we initialise the model with four clusters, canopy centroids initialisation and Manhattan distance. The number of clusters is four because there are four different area types in the training points (section 2.4) defined.

LVQ algorithm takes these parameters:

- number of clusters,
- the learning rate,
- number of epochs,

- `boolean` which indicates if the attributes are normalised before being used by the algorithm.

We initialise the algorithm with four clusters, for the same reason as above, 0.05 learning rate, 10 000 epochs and normalised attributes.

After training the clustering model we are able to cluster over any feature collection or geometry. However, to filter only those features which were clustered as croplands, we need to know the number of the croplands cluster.

To achieve that we cluster the training data. It results in having a cluster and area type properties on each feature. Once we filter only those features which have type equal to cropland we can iterate over them and check which cluster number is the most frequent. This idea is presented in Listing 7.1.

```
def find_field_cluster_asynchronous(
    field_collection: ee.FeatureCollection
) -> ee.Number:
    clusters = field_collection.iterate(
        lambda elem, prev: ee.List(prev).add(
            elem.get("cluster")
        ),
        ee.List([]),
    )
    return ee.Number(
        ee.List(clusters).reduce(ee.Reducer.mode())
    ).int8()
```

Listing 7.1: Finding the cropland cluster number

## 7.3 Feature and yield data association

Now that we can extract cropland features we have to associate them with the yield data.

We obtain the cropland features by clustering over the area of the given district. The corresponding clusterer is chosen for the currently processed year. As a response, we get a `ee.FeatureCollection` with features representing all the croplands in the district.

Because we have only one yield value for each district, we perform a mean aggregation over each band of the `ee.FeatureCollection` to get only one feature for a whole year. This feature is matched with a yield value.

To demonstrate how to associate the Landsat collection with the crop yield data, we provide a snippet with pseudocode of the idea.

This collection of associated values is an input of the prediction model. Since it is essential to have different data for training and testing, we split the

```
input  : $years$ – a list of years
input  : $area$ – the area of a district
output: a feature collection – one feature per each year

let features be an empty list;
foreach $y \in years$ do
    cropland-features ← cropland features in $area$ (use clustering);
    agg-data ← mean aggregation over each band in cropland-features;
    agg-data ← agg-data + correct yield for $y$ and $area$;
    features ← features ∪ agg-data;
end
return $features$
```

Algorithm 7.1: Creating a feature–yield collection

list of years into a training and testing one and build the association only for those years which are present in the requested list.

## 7.4 Prediction

In GEE, the classification and regression algorithms are located in the package `ee.Classifier`. There are fifteen different algorithms, some of them can be used only as classifiers, others only as regressors and some of them can be both.

Linear Regression is implemented under the name `gmoLinearRegression` and SVM under the name `svm`. The default prediction mode for SVM is classification, but we can manually override this setting and switch to regression by calling `svm.setOutputMode("REGRESSION")` method.

We initialise the Linear Regression with Lasso $L_1$ regularisation and parameter $\alpha = 0.5$. The gradient descent performs maximally two hundred iterations, and the algorithm uses logistic loss function for the regularisation instead of summing the coefficients.

SVM is of $\nu$-SVR type with RBF kernel and the parameters set as follows:

- $\gamma = 0.15$,
- $C = 100$,
- $\nu = 0.5$.

We create a unique prediction model for each of the districts. This model is trained on a *training feature collection* and tested on a different collection. The results achieved on the testing data are stored and returned for calculation of an overall model error. The only difference between the input to

```
{
    "type": "Feature",
    "geometry": null,
    "id": "0",
    "properties": {
        "blue": 27.763421292083713,
        "classification": 1.7263883352279663,
        "evi": 0.40007262735505134,
        "green": 21.27206551410373,
        "ndvi": 0.477380821225424,
        "near_infrared": 50.645131938125566,
        "red": 17.88535031847134,
        "shortwave_infrared_1": 39.06096451319381,
        "shortwave_infrared_2": 16.51410373066424,
        "yield_value": 1.73
    }
}
```

Listing 7.2: A classified feature

the `classify()` method and output is the property `classification` which contains the predicted value.

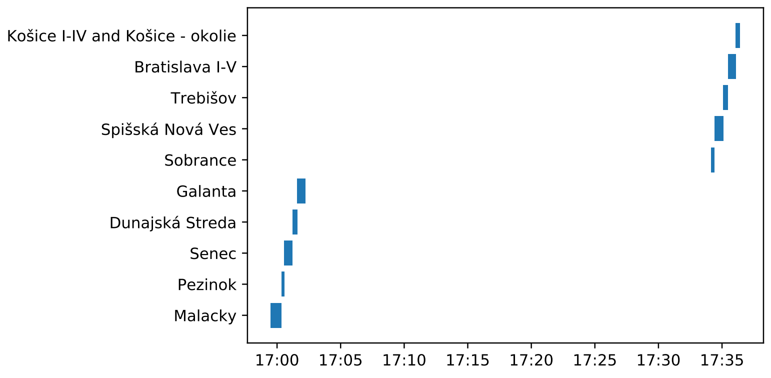
An input training feature collection is always a district-specific and contains one feature per each either training or testing year. Each feature consists of:

- a yield value for the given year and district, taken from past crop yields dataset,
- a mean band aggregation of all the features clustered as cropland in the given district.

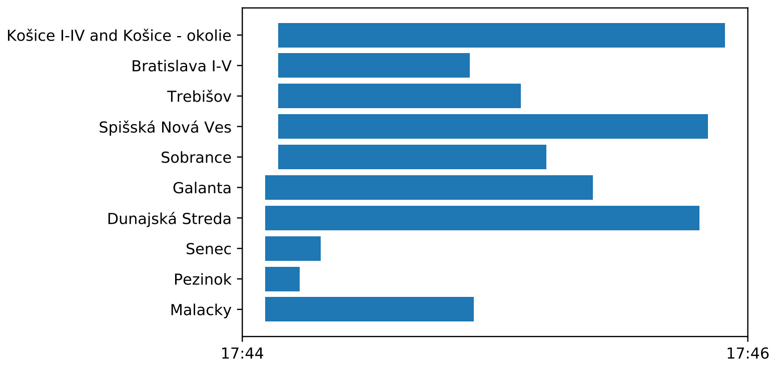
## 7.5 Parallelisation

To ensure the best performance possible, we try to postpone any call to the GEE API to the last moment so that we do not have to wait for the results to return.

The slowest part of the entire model is the prediction itself. Actually, it is the only place where we call `getInfo()` method and send all the commands to the server for evaluation. Once sent, the code execution is suspended on the client and waits until the server returns a result. Only then the execution is restored and proceeds to the next district. This process is repeated seventy-one times.



(a) Sequential computations



(b) Parallel computations

Figure 7.1: Sequential and parallel computations

As there are no dependencies between the prediction models, the whole routine is easily parallelisable. All we need to do is running a new thread for each of the districts. We run seventy-one threads on our local machine. Every one sends the data to a server and waits for the response which it appends to the results. It makes no computations and only sleeps for most of its runtime.

In Figure 7.1 we see how the two approaches differ. Both figures display the same districts (the first and the last five) and the times when the computations started and ended for them. The sequential computation for all seventy-one districts took 36 min 51 s while the parallel computation lasted only 3 min 54 s which is *twelve times faster*.

Listing 7.3 shows how the parallel algorithm is implemented. It uses Python's `threading` package. The `func` passed as a parameter to this function must take two arguments where the first argument is the district for which to run the computation and second is a `list` to which the function `appends` the results. There is no need for us to lock the access to the list because it is a responsibility of Python's `list` to implement the method as atomic.

```python
def iterate(self, func: Callable) -> List[Dict]:
    return_value, threads = [], []
    for region in self.regions.values():
        t = threading.Thread(
            target=func, args=(region, return_value)
        )
        t.start()
        threads.append(t)
    [t.join() for t in threads]
    return return_value
```

Listing 7.3: Parallel iteration over the districts

| **Cereals** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | 0.3673 t ha$^{-1}$ | 0.3394 t ha$^{-1}$ |
| **LVQ** | 0.4255 t ha$^{-1}$ | 0.2823 t ha$^{-1}$ |

(a) Cereals

| **Potatoes** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | 5.5228 t ha$^{-1}$ | 5.8331 t ha$^{-1}$ |
| **LVQ** | 5.9314 t ha$^{-1}$ | 5.7316 t ha$^{-1}$ |

(b) Potatoes

Table 7.1: MAE scores

## 7.6 Measured model errors

In the previous sections and chapters, we introduced the models, the algorithms and the procedures we perform in order to predict crop yield. In this section, we discuss how well the models work.

A separate test is run for each combination of a clustering and prediction algorithm. As we have two algorithms of each type, there are four tests in total for each of the observed crops. A test uses the whole range of nineteen years. This range is randomly split into training and testing years in a way that fourteen years are used for training and five for testing. The final MAE is computed from 355 predictions for the testing years over all the districts.

Table 7.1 presents the measured errors for each combination of clusterer and predictor on both data sources (cereals and potatoes). The fact that the best results are achieved by different models for the observed crops is unexpected. For cereals the best results are achieved by LVQ–SVM with MAE equal to 0.2823 t ha$^{-1}$, and by *k*-means – Linear Regression for potatoes with MAE 5.5228 t ha$^{-1}$. We consider both these results as very good given the simplicity of the models.

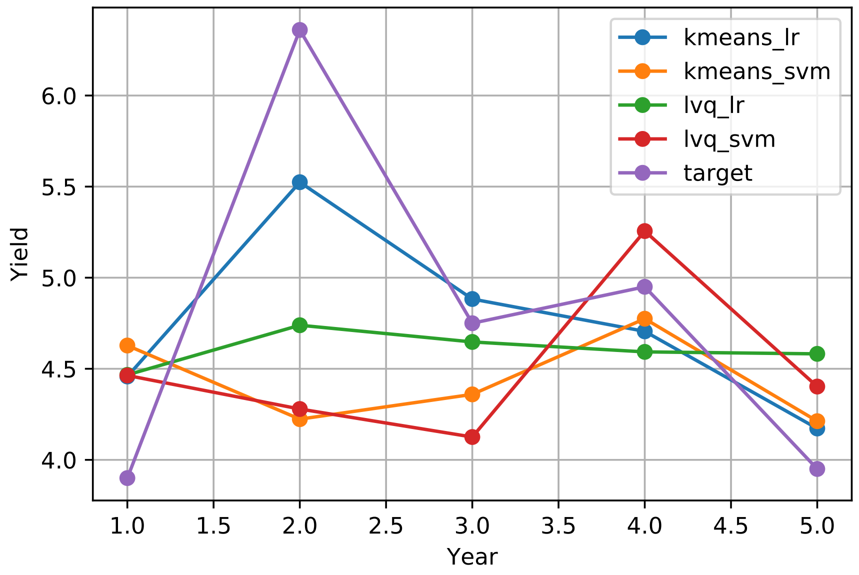


Figure 7.2: The models comparison in the District of Senec

| **Cereals** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | $0.4676\,\mathrm{t\,ha^{-1}}$ | $0.4877\,\mathrm{t\,ha^{-1}}$ |
| **LVQ** | $0.5137\,\mathrm{t\,ha^{-1}}$ | $0.3348\,\mathrm{t\,ha^{-1}}$ |

(a) Cereals

| **Potatoes** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | $6.6981\,\mathrm{t\,ha^{-1}}$ | $6.8531\,\mathrm{t\,ha^{-1}}$ |
| **LVQ** | $7.0453\,\mathrm{t\,ha^{-1}}$ | $6.7526\,\mathrm{t\,ha^{-1}}$ |

(b) Potatoes

Table 7.2: RMSE scores

Figure 7.2 compares the models on the cereals dataset in the district of Senec for the five testing years. It is clear that none of the models describes the target values precisely, but they provide a satisfactory estimation which differs from the target value only a little. Even though it may seem that the *k*-means – Linear Regression combination estimates the target function better than the other three it pretty well can be a case of this particular district and LVQ–SVM outperforms it in all the other districts.

In Table 7.2 we provide RMSE as well in order to keep compatibility with the related papers where they report the results using this loss function.

All in all, we consider the suggested prediction pipeline a success which provides a prediction which error is well beyond the standard deviations (section 2.3) of the observed crops.

Chapter 8

# Improvements

As we believe there is still some room for improvements left, we dedicate this chapter to suggesting improvements that can make the prediction even more precise. The first improvement to be mentioned has already been implemented and has played a role in the model already presented. The other two are only comments on which way the research could continue in the future, and the last one is an experiment which we have implemented, and we will present the achieved results.

## 8.1 EVI and NDVI computation

The first improvement which is already implemented concerns the vegetation indices computation. Formerly, the indices were calculated from the integer rounded values of the other bands which caused a notable error in the indices values. It was mainly EVI where we registered the problem. The index's values should belong to the interval $[-1, 1]$. A great many of the values did not fulfil this condition.

This issue was caused by the fact that for calculation we used integers instead of the original float values. To fix this, we create a new composite with enforced no integer conversion policy. We calculate the indices values on this composite and then copy the EVI and NDVI values back to the composite which has the band values converted to integers.

Figure 8.1 shows a series of EVI values when calculated from floats (blue) and integers (orange). It is caused by using multiplicative coefficients in the EVI equation (see subsection 2.1.2).

This issue does not affect NDVI index as it is only a normalised difference of two arbitrary numbers of $R$ with no other mathematical operations on it.

Implementing the better EVI and NDVI computation improved MAE by more than 0.1 on $k$-means – Linear Regression model for cereals.

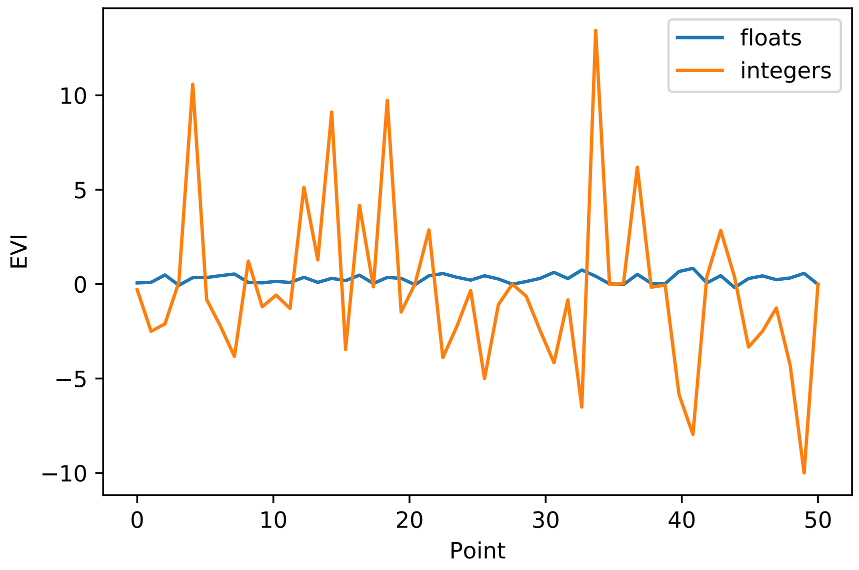


Figure 8.1: EVI values comparison

## 8.2 Date range

Currently, we are taking the remote sensing data from the whole year into account. This collection of images which starts on the 1st of January and ends on the 31st of December is aggregated into a single image composite. By doing this, we count in some values which are not that relevant and can affect the resulting image in an unwanted way.

For the mentioned reason we see a potential improvement in choosing a better date range to make the composite for. It should exclude the winter months and probably the first spring and the last autumn month as well. Still, this is only an assumption which would require some further research to establish this range.

Just a quick test on cereals showed that narrowing the date range down to 1st April – 31st October made the predictions less accurate. We achieved:

- MAE = 0.3367 for LVQ–SVM
- MAE = 0.32 for $k$-means – Linear Regression

## 8.3 Other vegetation indices

In this work we used only two vegetation indices; nonetheless, there are many others which we have not tried. It could be worth calculating and using some of them both in feature extraction and prediction process.

Lu et al. [14] list some other vegetation indices which we have not mentioned yet:

- Soil-Adjusted Vegetation Index (SAVI)
- Green-Red Ratio Vegetation Index (GRVI)
- Generalised Difference Vegetation Index (GDVI)

Each of the indices has its strengths and weaknesses. For example, GRVI is good at capturing seasonal differences in photosynthetic capacity, EVI performs well in monitoring activity across an ecosystem and SAVI is beneficiary if one needs to reduce the effect of soil brightness [14].

For this, it is vital to understand the indices and their capabilities to achieve the best results possible.

## 8.4 Lowland–Highland clustering

The idea behind this improvement is to split the districts into two parts by their geomorphology (if a district is situated on lowlands or highlands) and build an individual clusterer for lowlands and highlands. We expect that this helps the clusterer to recognise the patterns and similarities in the data better as we believe that some values may overlap in various parts of the country.

We assign a land type to a district according to its region by the region – land type mapping specified in Listing 8.1.

```
REGION_LAND_MAPPING = {
    "Bratislavský": Land.Lowland,
    "Trnavský": Land.Lowland,
    "Nitriansky": Land.Lowland,
    "Trenčiansky": Land.Highland,
    "Žilinský": Land.Highland,
    "Banskobystrický": Land.Highland,
    "Prešovský": Land.Highland,
    "Košický": Land.Lowland,
}
```

Listing 8.1: Region – land type mapping

For this improvement, we also created a new dataset used for training the clustering models. It contains information on the land type of training points.

We run the experiments here in the same way as in the previous chapter. We display MAEs of these tests in Table 8.1. By comparing these results with the original, we see that this improvement is no breakthrough in our models because we recorded improvement only in half of the models tested. To be

| **Cereals** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | $0.5136\,\mathrm{t\,ha^{-1}}$ | $0.2836\,\mathrm{t\,ha^{-1}}$ |
| **LVQ** | $0.4076\,\mathrm{t\,ha^{-1}}$ | $0.3274\,\mathrm{t\,ha^{-1}}$ |

(a) Cereals

| **Potatoes** | **Linear Regression** | **SVM** |
|---|---|---|
| ***k*-means** | $6.69\,\mathrm{t\,ha^{-1}}$ | $5.7871\,\mathrm{t\,ha^{-1}}$ |
| **LVQ** | $5.3114\,\mathrm{t\,ha^{-1}}$ | $5.7572\,\mathrm{t\,ha^{-1}}$ |

(b) Potatoes

Table 8.1: Improved model's MAE scores

exact, the two worse models from the baseline improved their MAEs and vice versa.

What we consider a notable improvement is the one of LVQ – Linear Regression which improved from the original $5.9314\,\mathrm{t\,ha^{-1}}$ to $5.3114\,\mathrm{t\,ha^{-1}}$ on the potatoes dataset.

Chapter 9

# Predictions for 2019

In this chapter, we discuss and analyse the predictions for 2019. For doing this, we use the two best models based on their MAE.

All parts (east, middle and west) of the Slovak Republic have at least one representative district amongst the presented subset. We also keep in mind the geomorphological variance of the selected districts.

To present predictions on a district level, we need to define the cropland areas of the selected districts and their shares in the cereals and potatoes total areas. Of the overall 1 407 729 ha cropland area in the Slovak Republic the selected districts apportion the areas as presented in Table 9.1 [29]. Under the assumption that the potato and cereal croplands are uniformly distributed amongst the districts, we calculate how much area these crops cover in a district. It will either confirm or reject a hypothesis about the adequacy of the predictions.

In 2017, potatoes were cultivated on an area of 7450 ha and for cereals it was 717 471 ha [30]. If a cropland area of a district $d$ is $S_d$, we calculate potato area by the formula:

$$S_{d\text{-}potatoes} = \frac{S_d}{1\,407\,729} 7450 \tag{9.1}$$

Now we use the areas described above to calculate the total yields in each of the selected districts. The results are presented in Table 9.2. Firstly, we

| District | Total area | Cereals area | Potato area |
|---|---|---|---|
| **Komárno** | 75 975 ha | 38 722 ha | 402 ha |
| **Michalovce** | 48 154 ha | 24 542 ha | 255 ha |
| **Poprad** | 11 460 ha | 5841 ha | 61 ha |
| **Trnava** | 48 278 ha | 24 606 ha | 255 ha |
| **Žilina** | 10 231 ha | 5214 ha | 54 ha |

Table 9.1: Cropland areas of the selected districts

| | **Cereals** | **Potatoes** |
|---|---|---|
| **Komárno** | 4.3371 t ha$^{-1}$ | 16.9471 t ha$^{-1}$ |
| **Michalovce** | 3.5754 t ha$^{-1}$ | 15.4930 t ha$^{-1}$ |
| **Poprad** | 3.1427 t ha$^{-1}$ | 22.9324 t ha$^{-1}$ |
| **Trnava** | 4.8515 t ha$^{-1}$ | 24.7483 t ha$^{-1}$ |
| **Žilina** | 2.7078 t ha$^{-1}$ | 25.0665 t ha$^{-1}$ |

(a) Per hectare yield predictions

| | **Cereals** | **Potatoes** |
|---|---|---|
| **Komárno** | 167 941 t | 6813 t |
| **Michalovce** | 87 747 t | 3950 t |
| **Poprad** | 18 357 t | 1399 t |
| **Trnava** | 119 376 t | 6311 t |
| **Žilina** | 14 118 t | 1354 t |

(b) Total yield prediction

Table 9.2: 2019 predictions for the selected districts

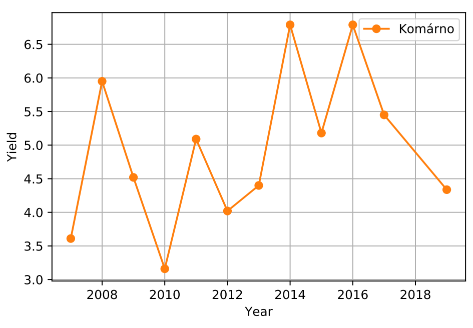
show the predicted t ha$^{-1}$ values, and then we multiply these values by the area and display the outcomes in the second table.

In order to provide a context to the predicted hectare yields, we plot time-series of the yields during the last ten years. These graphs enable us to conclude if the predicted yield is an average, exceptionally high or low. The graphs show the time series for both the crops for the two districts with the highest yields.

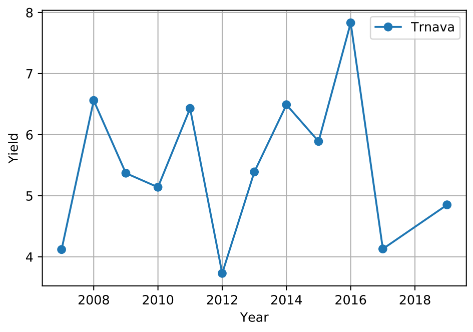
The cereals predictions are rather pessimistic and lower than they were in the last years. It is better for potatoes as the prediction for Žilina is the highest of all the yields achieved in the last ten years. Trnava was assigned an average yield for this year. None of the values shows outlaying which would indicate some noteworthy error in the prediction pipeline or in the methodic approach we applied in the process.

The cereal yields hardly make anyone wondering as the results fulfil our expectations and confirm that the predictions are realistic. That is because the lowland districts (Komárno and Trnava) achieved higher hectare yields and have significantly higher total yields which are caused by vaster cropland areas. The smaller hectare yields of the mountainous districts (Poprad and Žilina) only supports the idea of these predictions to be realistic.

What we consider unusual are the potato yields for the already mentioned highland districts. The predictions are equal to 22.9324 t ha$^{-1}$ for the District of Poprad and 25.0665 t ha$^{-1}$ for the District of Žilina which are higher than the lowland estimations. This peculiarity can be caused by the recent changes in the climate which cause higher summer temperatures and consecutive droughts in the south of the Slovak Republic which can cause the potatoes

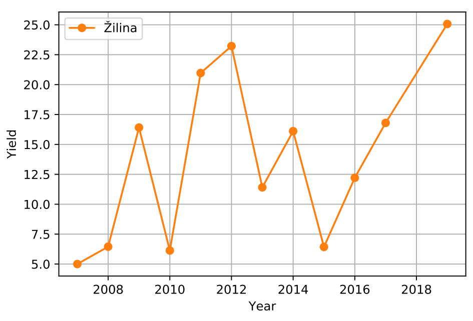


(a) The District of Komárno

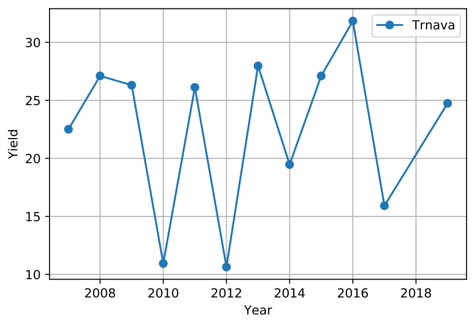


(b) The District of Trnava

Figure 9.1: Cereals 2007–2019 time-series



(a) The District of Žilina



(b) The District of Trnava

Figure 9.2: Potatoes 2007–2019 time-series

shortage and on the other hand excellent conditions for potatoes to grow in the north.

However, we have to emphasise the fact that the predictions were created by analysing only the first four months of the year which are not that informative as the months only to come. It would be interesting to run the prediction models in the middle of summer when the crops are in their vegetation period.

Chapter **10**

# Project environment

Our work comes with an implementation of the presented prediction pipeline. To ease the environment installation process and the usage of the software we guide a user through the process in this chapter.

The source code of our project is available either on GitHub (`https://github.com/ondrejpudis/crop-yield-predictor`) or on the enclosed CD.

## 10.1 Installation

Because each of us may use a different computer, operating system and configuration we decided to use the Docker technology and create an image based on `python:3.7-slim-stretch` which covers all the necessary dependencies and the only step a user is required to do is build the image. Another advantage of this approach is that no dependencies are installed into the user's computer; everything is isolated and virtualised. The full definition of our image is shown in Listing 10.1.

The user is required to have a functional installation of Docker on their machine and a copy of our project directory. If these requirements are satisfied the user can run the build command:

```
docker build -t yield-predictor .
```

It will download all the required layers and ensemble an image which will be tagged as `yield-predictor`.

After a successful build the user can create and run a new container:

```
docker run -i -t --name predictor yield-predictor bash
```

This command will create a new container named `predictor` and log the user into the terminal. Here the user has to authenticate themselves against the GEE service. It is done by running:

```
FROM python:3.7-slim-stretch

WORKDIR /app

ENV PACKAGE_VERSION=1.0

RUN pip install --upgrade pip && \
    apt-get update && \
    apt-get -y install openssl libssl-dev gcc

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app/

RUN pip install -e .
```

Listing 10.1: Docker image configuration

```
earthengine authenticate
```

Now the user can use the software without restrictions. The available commands and their parameters are listed in section 10.2.

After the user exits the container it is stopped, however still exists and can be restarted:

```
docker start predictor
```

and stopped:

```
docker stop predictor
```

For logging into the container's `bash` the user can use:

```
docker exec -i -t predictor bash
```

## 10.2 Command-line interface

There are two commands available in our project. During the build stage, they are installed and registered within the operating system. We can use them as regular `shell` commands. Running the commands from elsewhere than the `/app` folder will fail with `FileNotFoundError`. This happens because we use relative paths like `Path("datasets/crop_data/cereals.csv")` in our project.

`test-predict` command trains and tests the given combination of models and returns an overall MAE. The options for this command are:

- `-c` clustering algorithm (either `kmeans` or `lvq`)
- `-p` prediction algorithm (either `lr` for Linear Regression or `svm`)
- `--start` a year to start in, must be higher or equal to 1999 (defaults to 1999)
- `--end` a year to end in, must be lower or equal to 2017 (defaults to 2017)
- `--rmse` show RMSE score along with MAE

`predict` command trains the given combination of models on all the previous years, predicts a yield for a given year and returns a dictionary of the district – predicted yield pairs. The options for the command are:

- `-c` clustering algorithm (either `kmeans` or `lvq`)
- `-p` prediction algorithm (either `lr` for Linear Regression or `svm`)
- `--year` a year to make a prediction for, must be lower or equal to 2019 (defaults to 2019)
- `--district` a district name to make a prediction for, if `None` the prediction is made for all the districts (defaults to `None`)

The following options are available on both the commands:

- `--crop` crop to run the command for (either `cereals` or `potatoes`, defaults to `cereals`)
- `--split` use lowland–highland split in the computation
- `--force-server` forces a computation to run on a server even if there is a result of the same computation already stored offline

# Conclusion

In this work, we explored the remote sensing data and its contribution to the environmental science. We showed that there are indices like NDVI and EVI which help us identify the vegetation level in the data. Moreover, these indices are capable of separating the green areas like croplands or woods from the non-cropland once like paved areas in cities. We also enumerated various datasets used in this work and found out that the Statistical Office and the Geodetic Institute provide many of them with a detailed scope, in high quality and for free.

Throughout the work, we were using Google's geospatial platform Earth Engine which provided us with access to the Landsat raw remote sensing collection. Both the Python API client and the web-based interface were a great help; even though, it was not simple to get used to the lazy computation client-server paradigm.

To achieve the goals we had to combine information from more incompatible sources. It, in the majority of cases, required a preprocessing, like a conversion from a general XML format into GeoJSON or from Microsoft Excel files into a CSV file which is well computer readable.

Our work demonstrated a simple prediction pipeline which extracts desired knowledge from the remote sensing data, associates this data with another information to create an input for a mathematical prediction model. Even though no complex neural networks were used in this work, the best-achieved MAEs of our predictions are $0.2823\,\mathrm{t\,ha^{-1}}$ for cereals and $5.3114\,\mathrm{t\,ha^{-1}}$ for potatoes.

Besides the results of the base pipeline, we suggested a few more improvements which include better date range choice of processed satellite imagery, usage of other vegetation indices or another data preprocessing methods like lowland–highland split.

The lowland–highland preprocessing was implemented and tested, and it was successful on potatoes dataset where the LVQ – Linear Regression beat the $k$-means – Linear Regression model by improving MAE value by $0.2114\,\mathrm{t\,ha^{-1}}$.

The other suggestions require additional research which is beyond the scope of this work.

Finally, we took the best models for both potatoes and cereals and predicted the yields for 2019 which seem realistic to us, despite the fact they were made only by analysing the first four months of the year. For example, in the District of Trnava, we predict the yield of potatoes to $24.75\,\mathrm{t\,ha^{-1}}$ which is nothing exceptional for this district if we have a look at the values from the last ten years.

# Bibliography

1. SCHOWENGERDT, Robert A. The Nature of Remote Sensing. In: *Remote Sensing: Models and Methods for Image Processing*. 3rd ed. Burlington: Academic Press, 2007, pp. 1–10. ISBN 978-0-12-369407-2. Available from DOI: `10.1016/B978-012369407-2/50004-8`.
2. MORAIN, Stanley A. A Brief History of Remote Sensing Applications, with Emphasis on Landsat. In: *People and Pixels: Linking Remote Sensing and Social Science*. Washington, DC: The National Academies Press, 1998, pp. 28–50. Available from DOI: `10.17226/5963`.
3. BLASCHKE, Thomas. Object based image analysis for remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2010, vol. 65, pp. 2–16. ISSN 09242716. Available from DOI: `10.1016/j.isprsjprs.2009.06.004`.
4. *Case Studies: How Landsat Helps Us* [online]. 2019-04-25. National Aeronautics and Space Administration of the USA, 2019 [visited on 2019-04-28]. Available from: `https://landsat.gsfc.nasa.gov/how_landsat_helps/case-studies-2/`.
5. YOU, Jiaxuan; LI, Xiaocheng; LOW, Melvin; LOBELL, David; ERMON, Stefano. Deep Gaussian Process for Crop Yield Prediction Based on Remote Sensing Data. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. San Francisco, 2017, pp. 4559–4566. ISBN 978-1-57735-785-8. ISSN 2374-3468. Available also from: `https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14435/14067`.
6. SABINI, Mark; RUSAK, Gili; ROSS, Brad. *Understanding Satellite-Imagery-Based Crop Yield Predictions*. Stanford University, 2017. Available also from: `http://cs231n.stanford.edu/reports/2017/pdfs/555.pdf`.

7. SIDHU, Nanki; PEBESMA, Edzer; CÂMARA, Gilberto. Using Google Earth Engine to detect land cover change: Singapore as a use case. *European Journal of Remote Sensing*. 2018, vol. 51, pp. 486–500. Available from DOI: `10.1080/22797254.2018.1451782`.
8. XIONG, Jun; THENKABAIL, Prasad S.; GUMMA, Murali K.; TELUGUNTLA, Pardhasaradhi; POEHNELT, Justin; CONGALTON, Russell G.; YADAV, Kamini; THAU, David. Automated cropland mapping of continental Africa using Google Earth Engine cloud computing. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2017, vol. 126, pp. 225–244. ISSN 0924-2716. Available from DOI: `10.1016/j.isprsjprs.2017.01.019`.
9. *Global Croplands* [online]. USGS Flagstaff, 2019 [visited on 2019-04-29]. Available from: `https://www.croplands.org/`.
10. HUETE, Alfredo; DIDAN, Kamel; MIURA, Tomoaki; RODRIGUEZ, Edna P; GAO, Xiang; FERREIRA, Laerte G. Overview of the radiometric and biophysical performance of the MODIS vegetation indices. *Remote Sensing of Environment*. 2002, vol. 83, pp. 195–213. Available from DOI: `10.1016/S0034-4257(02)00096-2`.
11. *Introduction to the Google Earth Engine* [online]. 2019-01-29. Google [visited on 2019-04-04]. Available from: `https://developers.google.com/earth-engine/`.
12. *Landsat 7 Collection 1 Tier 1 Raw Scenes* [online]. United States Geological Survey and National Aeronautics and Space Administration, 2019 [visited on 2019-04-29]. Available from: `https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C01_T1`.
13. LOYD, Charlie. *Putting Landsat 8's Bands to Work* [online]. 2013-06-14. 2013 [visited on 2019-04-29]. Available from: `https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands/`.
14. LU, Linlin; KUENZER, Claudia; WANG, Cuizhen; GUO, Huadong; LI, Qingting. Evaluation of Three MODIS-Derived Vegetation Index Time Series for Dryland Vegetation Dynamics Monitoring. *Remote Sensing*. 2015, vol. 7, no. 6, pp. 7597–7614. ISSN 2072-4292. Available from DOI: `10.3390/rs70607597`.
15. *District administrative borders* [online]. 2019-02-25. Bratislava: Geodetic and Cartographic Institute of the Slovak Republic, 2019 [visited on 2019-04-28]. Available from: `https://www.geoportal.sk/sk/zbgis_smd/na-stiahnutie/`.
16. *Hectar Yields of Selected Agricultural Crops* [online]. Bratislava: Statistical Office of the Slovak Republic, 2019 [visited on 2019-04-29]. Available from: `http://datacube.statistics.sk/`.

17. GORELICK, Noel; HANCHER, Matt; DIXON, Mike; THAU, David; ILYUSHCHENKO, Simon; MOORE, Rebecca. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*. 2017, vol. 202, pp. 18–27. ISSN 0034-4257. Available from DOI: `10.1016/j.rse.2017.06.031`.

18. *Google Earth Engine API* [online]. 2019-04-29. Google [visited on 2019-04-29]. Available from: `https://developers.google.com/earth-engine/api_docs`.

19. KLOUDA, Karel; VAŠATA, Daniel; MALDONADO LOPEZ, Juan P. Hierarchické shlukování a algoritmus k-means. In: *Vytežovaní znalostí z dat (BI-VZD)* [online]. 2018-12-06. Prague: Faculty of Information Technology of CTU, 2018, pp. 32–42 [visited on 2019-04-04]. Available from: `https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-04-cs-handout.pdf`.

20. ARTHUR, David; VASSILVITSKII, Sergei. K-Means++: The Advantages of Careful Seeding. In: *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. New Orleans: SODA, 2007, vol. 8, pp. 1027–1035. ISBN 9780898716245. Available from DOI: `10.1145/1283383.1283494`.

21. MCCALLUM, Andrew; NIGAM, Kamal; UNGAR, Lyle H. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, pp. 169–178. Available from DOI: `10.1145/347090.347123`.

22. KOHONEN, Teuvo K. Learning Vector Quantization. In: *The handbook of brain theory and neural networks*. 2nd ed. Cambridge, Massachusetts: MIT Press, 2003, pp. 631–635. ISBN 0-262-01197-2.

23. KLOUDA, Karel; VAŠATA, Daniel; MALDONADO LOPEZ, Juan P. Lineární regrese. In: *Vytežovaní znalostí z dat (BI-VZD)* [online]. 2019-03-25. Prague: Faculty of Information Technology of CTU, 2019, pp. 62–69 [visited on 2019-04-04]. Available from: `https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-07-cs-handout.pdf`.

24. KLOUDA, Karel; VAŠATA, Daniel; MALDONADO LOPEZ, Juan P. Regularizace a hřebenová regrese. In: *Vytežovaní znalostí z dat (BI-VZD)* [online]. 2018-12-06. Prague: Faculty of Information Technology of CTU, 2019, pp. 79–87 [visited on 2019-05-03]. Available from: `https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-09-cs-handout.pdf`.

25. BURGES, Christopher J.C. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*. 1998, vol. 2, no. 2, pp. 121–167. ISSN 13845810. Available from DOI: `10.1023/A:1009715923555`.

26. SMOLA, Alex J.; SCHOLKOPF, Bernhard. A tutorial on support vector regression. *Statistics and Computing*. 2004, vol. 14, pp. 199–222. Available also from: `http://alex.smola.org/papers/2004/SmoSch04.pdf`.

27. SMOLA, Alex J.; WILLIAMSON, Robert C.; BARTLETT, Peter L. New Support Vector Algorithms. *Neural Computation*. 2000, vol. 12, pp. 1207–1245. Available also from: `http://alex.smola.org/papers/2000/SchSmoWilBar00.pdf`.

28. WILLMOTT, Cort J.; MATSUURA, Kenji. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*. 2005, vol. 30, pp. 79–82. ISSN 0936-577X. Available from DOI: `10.3354/cr030079`.

29. *Štatistická ročenka o pôdnom fonde v SR*. Bratislava: Geodetic and Cartographic Institute of the Slovak Republic, 2019. ISBN 978-80-89831-08-1. Available also from: `http://www.skgeodesy.sk/files/slovensky/ugkk/kataster-nehnutelnosti/sumarne-udaje-katastra-podnom-fonde/statisticka-rocenka-2018.pdf`.

30. *Yields of selected agricultural crops* [online]. Bratislava: Statistical Office of the Slovak Republic, 2019 [visited on 2019-04-27]. Available from: `http://datacube.statistics.sk/`.

# Appendix A

# Glossary

**Docker** A containering platform to create isolated applications which are not dependant on the local machine architecture.

**GeoJSON** A format for encoding geographical data. The official specification is available at `https://tools.ietf.org/html/rfc7946`.

**Landsat** A system of satellites maintained by a joint programme of NASA and USGS [12].

**Python Package Index** A Python language software repository.

Appendix **B**

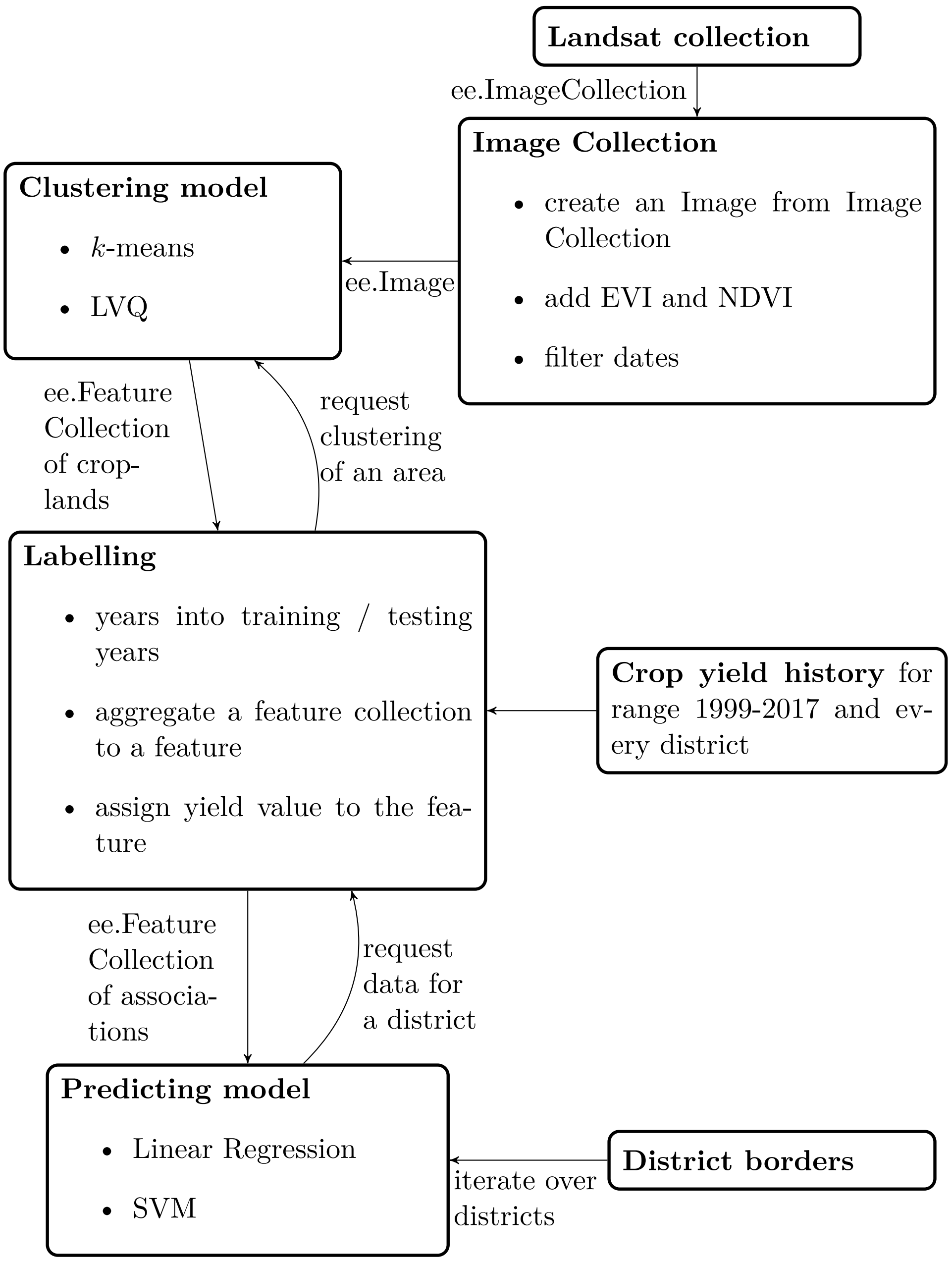
# Acronyms

|  |  |
|---|---|
| **API** | Application Programming Interface. |
| **CSV** | Comma Separated Values. |
| **ESA** | European Space Agency. |
| **Eurostat** | European Statistical Office. |
| **EVI** | Enhanced Vegetation Index. |
| **GDB** | Geodatabase. |
| **GEE** | Google Earth Engine. |
| **GRVI** | Green-Red Ratio Vegetation Index. |
| **IDE** | Integrated Development Environment. |
| **JSON** | JavaScript Object Notation. |
| **LVQ** | Learning Vector Quantization. |
| **MAE** | Mean Absolute Error. |
| **MODIS** | Moderate Resolution Imaging Spectroradiometer. |
| **MSS** | Multi Spectral Scanner. |
| **NASA** | National Aeronautics and Space Administration. |
| **NDVI** | Normalised Difference Vegetation Index. |

| | |
|---|---|
| **RBF** | Radial Basis Function. |
| **RMSE** | Root Mean Squared Error. |
| **SAVI** | Soil-Adjusted Vegetation Index. |
| **SVM** | Support Vector Machine. |
| **UN** | United Nations. |
| **USGS** | United States Geological Survey. |
| **XML** | Extensible Markup Language. |

# Appendix C

# Workflow diagram


Landsat collection
ee.ImageCollection
Image Collection
• create an Image from Image Collection
• add EVI and NDVI
• filter dates
ee.Image
Clustering model
• $k$-means
• LVQ
ee.Feature Collection of crop-lands
request clustering of an area
Labelling
• years into training / testing years
• aggregate a feature collection to a feature
• assign yield value to the feature
Crop yield history for range 1999-2017 and every district
ee.Feature Collection of associations
request data for a district
Predicting model
• Linear Regression
• SVM
iterate over districts
District borders

Appendix D

# Landsat versions

| Landsat version | Operational years | Spectral bands | Thermal bands | Resolution |
|---|---|---|---|---|
| Landsat I-V MSS | 1972-1978<br>1975-1982<br>1978-1983<br>1982-1993<br>1984-2012 | green, red, near infrared 1,<br>near infrared 2 | - | 60 m<br>30 m |
| Landsat IV-V TM | 1982-1993<br>1984-2012 | blue, green, red,<br>near infrared,<br>shortwave infrared | thermal infrared | 30 m |
| Landsat 6 | never reached the orbit | - | - | - |
| Landsat 7 | 1999-present | blue, green, red,<br>near infrared,<br>shortwave infrared,<br>panchromatic | low-gain thermal infrared<br>high-gain thermal infrared | 30 m<br>30 m<br>30 m<br>15 m |
| Landsat 8 | 2013-present | coastal aerosol,<br>blue, green, red,<br>near infrared,<br>shortwave infrared,<br>panchromatic<br>cirrus | thermal infrared | 30 m<br>30 m<br>30 m<br>30 m<br>15 m<br>15 m |

Table D.1: Landsat versions

Appendix E

# XML to GeoJSON conversion

```
import csv
import json
from pathlib import Path
import sys

from lxml import etree

csv.field_size_limit(sys.maxsize)

with open(
    Path("region_borders_parsed.csv"), "w"
) as destination:
    with open(Path("region_borders.csv"), "r") as source:
        source_dict = csv.DictReader(source)
        destination_dict = csv.DictWriter(
            destination, fieldnames=source_dict.fieldnames
        )
        destination_dict.writeheader()
        for r in source_dict:
            coordinates = (
                etree.fromstring(r["geometry"])
                .xpath(
                    (
                        "/Polygon/outerBoundaryIs/"
                        "LinearRing/coordinates"
                    )
                )[0]
                .text
            )
```

```
r["geometry"] = json.dumps(
    {
        "type": "Polygon",
        "coordinates": [
            list(
                reversed(
                    [
                        [
                            float(c)
                            for c in pair.split(
                                ","
                            )
                        ]
                        for pair in coordinates
                        .split(
                            " "
                        )
                    ]
                )
            )
        ],
    }
)
destination_dict.writerow(r)
```

Appendix F

# Contents of enclosed CD

```
Dockerfile ............ definition of project's Docker image
├── README.md ......... a project's description and installation guidelines
├── requirements.in ... a list of requirements of our project
├── requirements.txt .. compiled requirements, including their dependencies
├── setup.py .......... Python projec installation file
├── datasets .......... datasets used in the project
│   ├── crop_data ..... crop yields data in CSV format
│   └── geo_data ...... geographical data (district borders, training points)
├── javascript ........ scripts used for creating GEE visualisations
├── json .............. stored dumps of results returned from GEE servers
├── jupyter ........... notebooks with graph visualisations
│   └── pdf ........... exported PDFs from Jupyter notebooks
├── src ............... source codes of the project
└── text .............. the thesis text directory
    ├── source ........ source files of the thesis text
    └── thesis.pdf .... the thesis in PDF format
```