Department of Informatics, University of Zürich

Informatik-Vertiefung

Visualization of the Spatial Feed Data with Color Plots

Andrin Betschart

- -

Email: andrin.betschart@uzh.ch

25. März 2012

supervised by: Prof. Dr. Michael Böhlen
Dr. Andrej Taliun

1 Problem

1.1 Data

The Swiss Feed Database holds measurements of nutrients from feed samples collected from all parts of Switzerland. The origin of each sample is stored not only as city name but also as longitude and latitude in the database. Besides that the measured nutrients of each feed sample are stored as well.

1.2 Visualize

In the current on-line version, the feed samples are graphically illustrated as flags on the map and their containment of nutrients are given as text in the list. There is an essential drawback while visualizing feed samples with flags. If there are many feed samples from the same location, all of them are visualized by a single flag and, therefore, it is hard for the user to compare different locations based on the number of feed samples.
For example if we want to see from which location the most samples of the minerals potassium, magnesium and manganese come from, this can not easily be seen with the current implementation. See Figure 1.1 that illustrates this example and how it is solved.

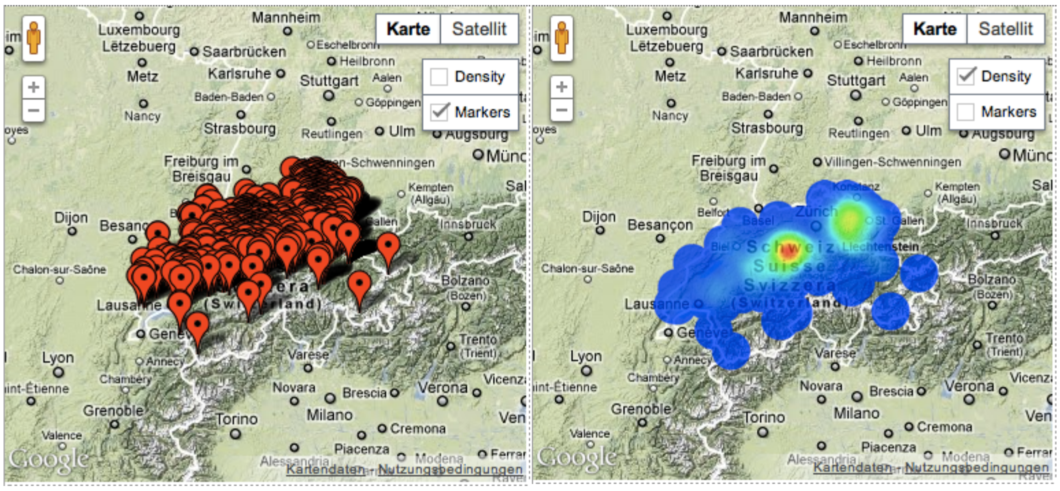


Figure 1.1 Visualization of the samples

The goal of this project is visualize the selected data samples on a map such that it is easy to recognize where the density of samples is high and where it is low. Chapter 2 of this report gives an introduction to the theory that is used to calculate the density of the samples. Then chapter 3 illustrates step by step how the implementation is done and chapter 4 gives a conclusion and suggests some further improvements, which might be done.

2 Kernel Density Estimation

The kernel density estimation is a method to compute the density of data samples in a smooth way. As input a finite data sample is given and the aim of the kernel density estimation is it, to estimate the density of the whole population at every point.

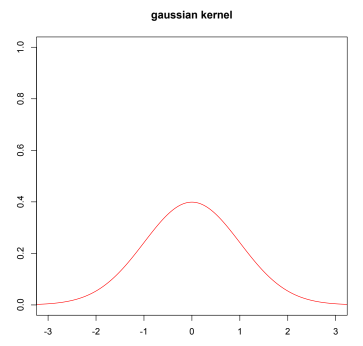
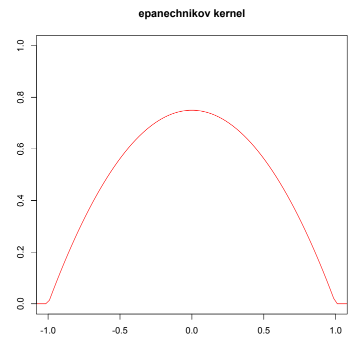
Let $X = \{X_1, \dots, X_n\}$ be a finite data sample from an unknown probability density function f. To estimate the shape of this function f, the kernel density estimator

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - X_i}{h})$$

with kernel function K and bandwidth, also called smoothing parameter, h can be used. The kernel function K is a symmetric function that satisfies the condition

$$\int_{-\infty}^{\infty} K(x)dx = 1.$$

Two popular examples are shown in the Table below.

Gaussian	$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$	
Epanechnikov	$K(x) = \begin{cases} \frac{3}{4}(1 - x^2), if \ \lvert x \rvert \ \leq 1 \\ 0, \qquad otherwise \end{cases}$	

In section 2.3 the selection of the best kernel function is shown, in order to minimize the computational effort, which is needed to compute the kernel density estimation.
Figure 2.1 shows an example of Kernel Density Estimation. A data set of 6 points $(X_1 = -2.1, X_2 = -1.3, X_3 = -0.4, X_4 = 1.9, X_5 = 5.1, X_6 = 6.2)$ is taken as input. Over every data point X_i a Gaussian Kernel with standard deviation 1 is placed. The red lines show those kernel functions. Finally the kernels are summed up to build the Kernel Density Estimate, which is illustrated by the blue curve.

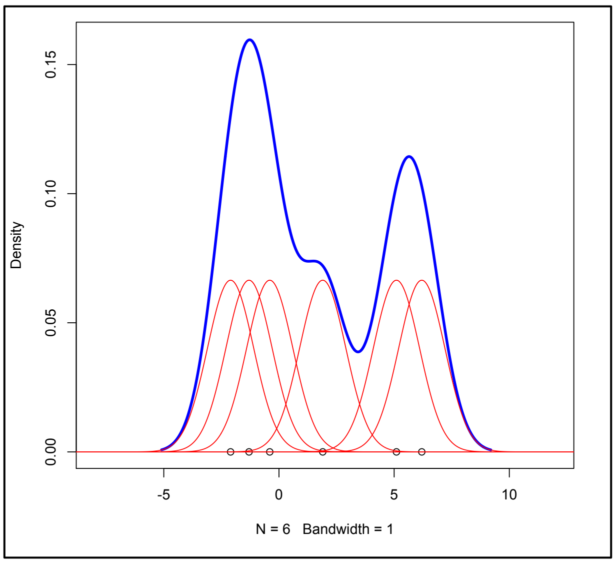


Figure 2.1 Kernel Density Estimate showing individual kernels with Gaussian kernel and bandwidth h = 1.

2.1 Bivariate Kernel Density Estimation

Until now we just looked at the Kernel Density Estimation for univariate data, but because it is the goal to visualize densities on maps we need to consider bivariate data. The definition of the Kernel Density Estimation as a sum of 'bumps' centered at each data point can easily be transformed to the bivariate case. The Bivariate Kernel Density Estimator is defined as

$$\hat{f}_h(x,y) = \frac{1}{nh^2} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}, \frac{y - Y_i}{h}\right)$$

with kernel function K and bandwidth h. Usually the kernel function K is a radially symmetric unimodal probability function.

For example the standard bivariate normal density function

$$K(x,y) = \frac{1}{2\pi}\, e^{-\frac{x^2+y^2}{2}}$$

or the bivariate Epanechnikov kernel

$$K(x,y) = \begin{cases} \frac{2}{\pi}(1 - x^2 - y^2)\,, & if\ x^2 + y^2 < 1 \\ 0\,, & otherwise. \end{cases}$$

Figure 2.2 illustrates the Bivariate Kernel Density Estimate of a data set of 5 points $(X_1 = \binom{2}{2}, X_2 = \binom{3}{2}, X_3 = \binom{6}{3}, X_4 = \binom{7}{7}, X_5 = \binom{8}{8})$.

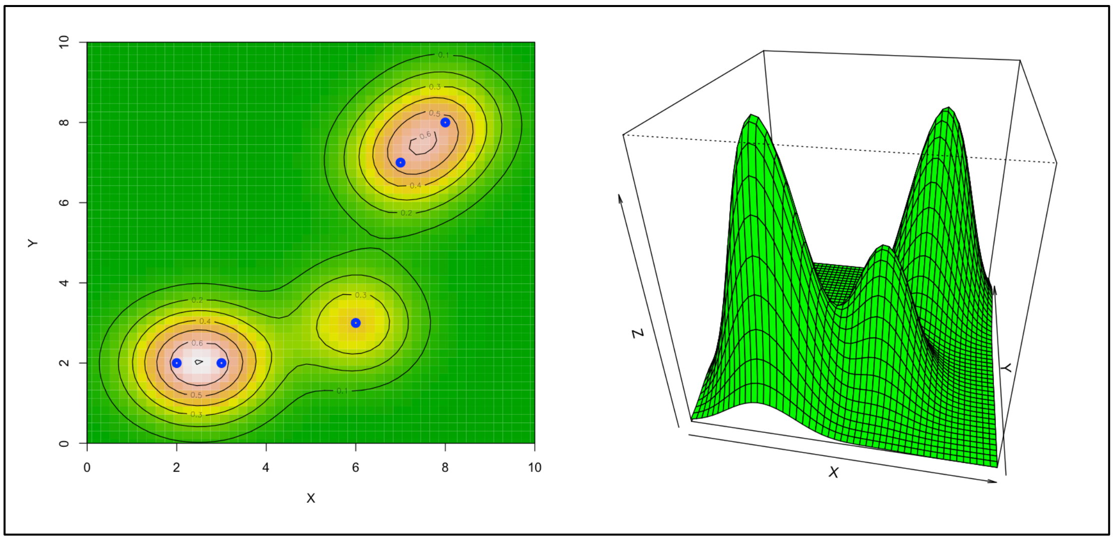


Figure 2.2 Bivariate Kernel Density Estimate with Gaussian Kernel and bandwidth h = 1 shown as colored 2-dimensional model with level curves (left) and as 3-dimensinal model (right).

2.2 Calculating the optimal bandwidth h

With choosing the optimal bandwidth h the deviation of the Kernel Density Estimate, of the given sample, from the true density of the whole population can be minimized. This can be achieved by calculating h_{opt} as

$$h_{opt} = \sigma * A(K) * n^{-\frac{1}{6}}$$

where σ is the variance of the data sample and $A(K)$ a constant that depends on the kernel. For the bivariate Gaussian Kernel $A(K) = 0.96$ and for the bivariate Epanechnikov Kernel $A(K) = 1.77$.

2.3 Computational considerations

Especially when the number of samples is large, it is important to take some care about what algorithms to use to calculate the Kernel Density Estimate. Choosing the right kernel function can lead to an efficient reduction of the computation time. For example if there are 1000 samples and the standard Kernel Density Estimator Function is chosen to calculate the density at each point of a grid with 1000 points, then nearly a million evaluations of the kernel function will be required. In this case it can make a noticeable difference if some arithmetic operations, for the calculation of the kernel function, can be omitted. Therefore it is not recommended to use the Gaussian Kernel, which need to evaluate the exponential function in every call. The Epanechnikov Kernel is under these circumstances a much better choice.

Another technique to optimize the calculation of the Kernel Density Estimate is explained in detail in chapter 3. The idea is to not calculate the density in every point of the grid, but to calculate the contribution of each data point to the points of the grid.

3 Implementation

The implementation of the color plots will directly be integrated in the currently available map. The idea is, to place a 'canvas' element of HTML directly into the overlay, which is provided by Google Maps API. This 'canvas' element then dynamically will be colored, based on the query results, by using the JavaScript language.
The following sections describe the steps that need to be implemented.

3.1 Placing the 'canvas' element on the map

As first step, the ,canvas' element, that displays the densities, needs to be placed on the map. This is done using an OverlayView of the Google Maps API. With the OverlayView one can put Objects, which are tied to latitude/longitude coordinates, on the map, like that the Objects move when dragging or zooming the map.

```
function DensityOverlay(map){
   this.canvas = null; this.div = null;
   this.minlat = 0, this.minlng = 0, this.maxlat = 0, this.maxlng = 0;
   this.setMap(map);
}

DensityOverlay.prototype = new google.maps.OverlayView();

DensityOverlay.prototype.onAdd = function() { }

DensityOverlay.prototype.onRemove = function() { }

DensityOverlay.prototype.draw = function()
{
   var overlayProjection = this.getProjection(),
   topleft = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.maxlat, this.minlng)),
   bottomright = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.minlat, this.maxlng));

   this.div.style.left = (topleft.x) + 'px';
   this.div.style.top = (topleft.y) + 'px';
   this.div.style.width = (bottomright.x - topleft.x) + 'px';
   this.div.style.height = (bottomright.y - topleft.y) + 'px';
}

DensityOverlay.prototype.setDataSet = function(data)
{
   //calculate maximum and minimum longitudes/latitudes

   var overlayProjection = this.getProjection(),
   topleft = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.maxlat, this.minlng)),
   bottomright = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.minlat, this.maxlng));

   this.div = document.createElement('DIV');
   this.div.style.position = "absolute";

   this.canvas = document.createElement("canvas");
   this.canvas.style.width = "100%";
   this.canvas.style.height = "100%";

   this.canvas.width = 1000;
   this.canvas.height = 1000;

   //calculate the density and color the ‘canvas’ element

   this.div.appendChild(this.canvas);

   var panes = this.getPanes();
   panes.overlayLayer.appendChild(this.div);
}
```

Example Code 3.1

The Example Code 3.1 shows how the 'canvas' element is placed on the map. By setting the *DensityOverlay*'s prototype to a new instance of *google.maps.OverlayView()* it gets an "subclass" of the Google Maps API Overlay class. In the constructor of the *DensityOverlay* the map, on which the layer is placed, is set. When the map is ready for the overlay to be attached and all data points are retrieved from the database the *setDataSet()* method is called. Within this method the properties of the 'canvas' element are loaded and it is attached to the 'div' element. And this 'div' element is finally appended as child to the *overlayLayer*. The *draw()* method is called when the map gets zoomed and when the overlay gets first displayed. Therefore the position of the 'div' element, that contains the 'canvas' element, is recalculated within this method, because the pixel position of the same coordinate change when zooming the map.

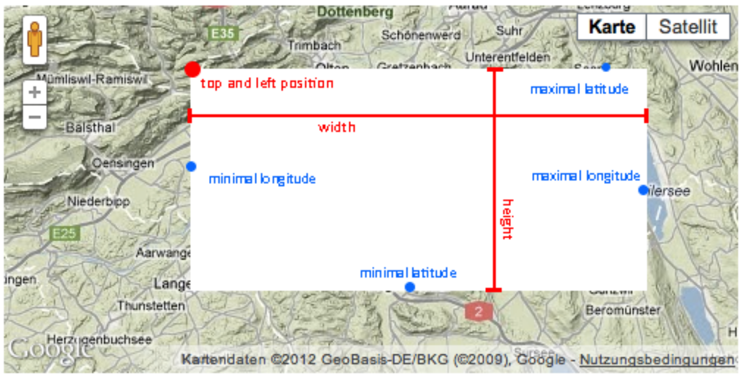


Figure 3.1 Describes the calculation of position and dimension of the 'canvas' element.

To determine the area over which the 'canvas' element must reach, the maximum and minimum latitudes respectively longitudes of the data points are searched. Those data points are transformed to pixels on the map. With these points the position and the dimension of the 'canvas' element can be calculated as shown in Figure 3.1. Because we are going to visualize the density of the data points by using the Kernel Density Estimation, we need to enlarge the 'canvas' element by *h* in every direction, this will be done immediately after calculating the optimal bandwidth *h* (See Section 3.3).

3.2 Transforming the data points

Now that we know the position and dimension of the ‚canvas' element we can transform the data points to pixels of the ‚canvas' element. After this transformation every data point has its x and y coordinate on the 'canvas' element.

The result queries from the database deliver the latitude and longitude of each feed sample. To display those on the map, they need to be projected on the overlay layer. For this matters the Google Maps API provides the method *fromLatLngToDivPixel*, with which the position on the overlay layer can be determinate for each latitude/longitude coordinate.

```
var data <- data points from the database.
var overlayProjection = this.getProjection(), points = new Array();
var topleft = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.maxlat, this.minlng)),
bottomright = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(this.minlat, this.maxlng));

for (var i = 0; i < data.length; i++)
{
    var p = overlayProjection.fromLatLngToDivPixel(new google.maps.LatLng(data[i].lat, data[i].lng))
    var point = {
        x: Math.floor((p.x - topleft.x) / (bottomright.x - topleft.x) * (this.canvas.width)),
        y: Math.floor((p.y - topleft.y) / (bottomright.y - topleft.y) * (this.canvas.height))
    };
    points.push(point);
}
```
Example Code 3.2

In the Example Code 3.2 the *points* array finally holds the x and y position on the 'canvas' element, of all data points (*data*).

3.3 Calculation of the density

First of all the kernel function must be chosen. The Epanechnikov Kernel has two important advantages, thus it is taken for the implementation. The first advantage it has is, that it can be computed quite fast. The second is, that it satisfies $K(x, y) = 0\ if\ x^2 + y^2 \geq 1$, what is necessary to compute the density with one scan of the data points. This will be explained below.

In order to get a correct value of density at every pixel, the calculation of the optimal bandwidth h needs do be done first. As we saw in chapter 2, the bandwidth is dependent on the chosen kernel function, the variance of the data points and the number of data points. Example code 3.3 shows how it is calculated in the implementation.

```
var points ← array containing x and y coordinates of all data points.
var xweight = 0, xqweight = 0, yweight = 0, yqweight = 0;
for (var p = 0; p < points.length; p++)
{
    var point = points[p];
    xweight += point.x;
    yweight += point.y;
    xqweight += Math.pow(point.x, 2);
    yqweight += Math.pow(point.y, 2);
}
var sx = (xqweight - (xweight * xweight / points.length)) / (points.length - 1);
var sy = (yqweight - (yweight * yweight / points.length)) / (points.length - 1);
var sigma = Math.sqrt( (sx + sy) /2);

var hopt =  Math.floor(sigma* 1.77 * Math.pow(points.length, -1/6));
```
Example Code 3.3

Once we have the optimal bandwidth we need to enlarge the 'canvas' element by h_{opt} in every direction. So that also the contribution of the data points at the borders of the 'canvas' element can be calculated. The enlargement by h_{opt} is, because every data point contributes to the density of all pixels that lay at most h_{opt} away from the data point itself, that is because we chose the Epanechnikov Kernel, which satisfies $K(x, y) = 0\ if\ x^2 + y^2 \geq 1$.

After that the estimation of the density in each pixel can be done. For every pixel the density is stored in a two-dimensional array. At the beginning this array only contains zeros. Now to calculate the density we iterate through all data points. For every pixel that is close enough (not more than h_{opt} pixels away) to the data point, the value in the array is increased by the contribution the current data point has on the density of the pixel. When finally iterated through all data points, the array contains the density value of every pixel. Also the maximum density value is stored, because it will be needed to color the 'canvas' element. The Example Code 3.4 shows the calculation of the density.

```
var points ← array containing x and y coordinates of all data points.
Var hopt ← calculated optimal bandwidth.

function epanechnikov_kernel(x,y)
```

```
{
    var t = Math.pow(x,2) + Math.pow(y,2);
    if (t >= 1)
        return 0;
    return 2 / Math.PI * ( 1 - t);
}

var densities = new Array();
var max = 0;

for (var i = 0; i < this.canvas.height; i++)
{
    var dd = new Array();
    for(var j = 0; j < this.canvas.width; j++)
        dd.push(0);
    densities.push(dd);
}

for (var p = 0; p < points.length; p++)
{
    var point = points[p];
    for (var i = point.y - hopt; i <= this.canvas.height && i <= point.y + hopt; i++)
    {
        for (var j = point.x - hopt; j <= this.canvas.width && j <= point.x + hopt; j++)
        {
            if (i > 0 && j> 0)
            {
                var new_density = densities[i-1][j-1] + epanechnikov_kernel((point.x-j)/ hopt,
                    (point.y-i)/ hopt);
                densities[i-1][j-1] = new_density;
                if (new_density > max)
                   max = new_density;
            }
        }
    }
}
```

Example Code 3.4

With this method, the calculation of the density can be made with one scan of the data points. This can be done because we chose the Epanechnikov Kernel, with which every data point only has an influence on the pixels that lay very close. If calculating the density pixel after pixel, the computation took much longer, because for every pixel all data points needed to be scanned.

3.4 Coloring the 'canvas' element

Once the contribution of all data points have been summed up in the array, the coloring of the 'canvas' element can be done. Therefore a color palette is needed. In order to provide an intuitive understanding of the visualized data, a common color palette is used. This color palette is shown in Figure 3.2. The pixel with the highest density (maximum value in the array) is colored red and all other pixels are colored according to their value in the array.

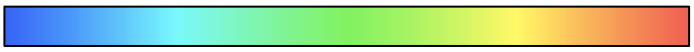


Figure 3.2 Color palette used to visualize the densities.

The Example Code below shows, how the coloring of the pixels on the ‚canvas' element is done in the implementation.

```
var densities ← array containing the density values of every pixel.
Var max ← the maximal density value.
var gradient ← array containing the color values oft he color platte.
var opacity ← the maximal opacity to use.
var ctx = this.canvas.getContext("2d"),
    pixels = ctx.createImageData(this.canvas.width, this.canvas.height);

for (var i = 0; i < this.canvas.height; i++) {
    for (var j = 0; j < this.canvas.width; j++) {
        var p = (i*this.canvas.width + j)*4;

        var v = Math.ceil(densities[i][j] / max * 255);

        pixels.data[p]= gradient[v*4];
        pixels.data[p+1]= gradient[v*4+1];
        pixels.data[p+2]= gradient[v*4+2];
        pixels.data[p+3] = (v==0) ? 0 : opacity;

    }
}
ctx.putImageData(pixels, 0, 0);
```
Example Code 3.5

3.5 Result

To see the outcome of the implementation we look at an example. The Figure below shows, from which location the most samples of the minerals potassium, magnesium and manganese come from. It can be seen, that a lot of samples come from the region around Willisau and another peak lies between Zürich and St. Gallen. And only few samples come from the parts that are colored blue.

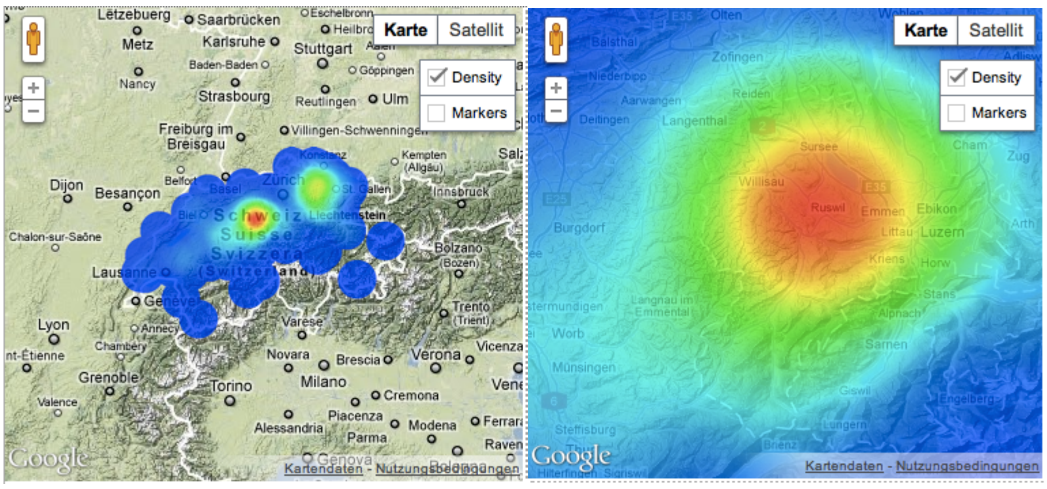


Figure 3.3 Shows an example in standard view (left) and zoomed in (right).

4 Conclusion

4.1 Evaluation

The evaluation of the implementation shows that the duration of the coloring grows linear with the number of data points. This is actually a good thing. But unfortunately with a lot of data points the computation anyhow takes some seconds. This might be the case, because the whole calculation is done on one machine. The evaluation was done with a set of random locations within Switzerland taken from the database. And the 'canvas' element was set to 1000 pixels in each dimension. With less than 10'000 data points the duration of the coloring is beyond 30 seconds and goes up to something more than 8 minutes for 100'000 data points.

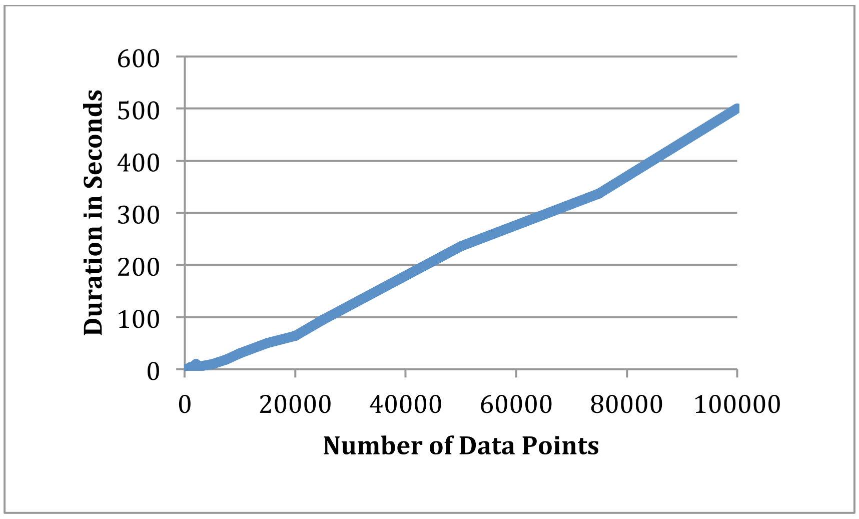


Figure 4.1 Shows the duration the coloring takes dependent on the number of data points.

4.2 Further improvements

At the moment, the implementation is not very fast for a lot of data points. In order to make the calculation of the density and the coloring faster it might be an idea to process those steps on the server. The server could calculate a density image and send this to the client. The client then only needs to put the image onto the map.

Another improvement that could be made is, to also visualize the containment of the selected nutrients on the map. At the moment only the number of samples can be visualized, but maybe someone wants to see where the quantity of magnesium in the samples is high and where it is low.

An advantage of this implementation is, that the calculation of the density only needs to be done when the data points change, but not when the user drags or zooms the map. But contrariwise, if the data points are spread over a wide area of the map, then zooming into the map does only show the global density and not the density of the seen part of the map. Therefore it might be an idea to recalculate the density for only the displayed area after zooming in or out of the map. Like that the distribution of the data points could be visualized in more detail when zooming in.

5 References

- Härdle, Müller, Sperlich, Werwarz (1995). *Nonparametric and Semiparametric Models.* Springer, Berlin.
- Scott, D.W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Wiley & Sons, New York.
- Silverman, B.W. (1986). *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London.