Online Computation of Up-To-Date Summaries in the Swiss Feed Database

Bachelorarbeit im Fach Informatik

vorgelegt von

Samuele Zoppi

Zürich, Schweiz

Angefertigt am

Institut für Informatik

der Universität Zürich

Prof. Dr. M. Böhlen

Betreuer: Dr. Andrej Taliun

Abgabe der Arbeit: 25.08.2011

Acknowledgments

My most sincere thanks goes to Dr. Andrej Taliun, who guided and supported me through the entire work. His explanations and comments contributed in a remarkable way to the success of the project and furthermore to develop myself intellectually. I am extremely grateful to Prof. Michael Böhlen, who gave me the opportunity to work on a very interesting and stimulating project - I really appreciated the possibility to contribute to the development of a real database.

Many thanks to Annelies Bracher of the Swiss Agronomic Institute (Agroscope), who explained the work of the researchers at her laboratories to me and made me discover the complex and fascinating world of animal breeding. Many thanks also to Francesco Cafagna, PhD student, who helped me going on with my work, answering a lot of questions and always having patience with me - it was very nice to have him in the team.

I would like to thank my family, which always encouraged me and believed in me. Finally, a very special thanks goes to my friends Frida Juldaschewa and Francesco Luminati, who helped and supported me during this work giving me a lot of precious advice - it is such as these little but constant contributions that make working easier and nicer.

Abstract

This thesis develops the up-to-date summary, an automatic approach to aggregate histories of nutrient measurements in the Swiss Feed Database. Since measurements are taken irregularly and are sparse in the time, a simple aggregation over the entire history is not representative of the real world state. We fight this challenge by detecting trends in history of measurements with a set of data fitting functions: uniform fitting function, linear regression and kernel regression. The experimental evaluation proves the scalability of our approach to aggregate the measurements of good and bad quality data. Further, this thesis contributes to the development of the Feed Database with the integration of the up-to-date summaries into web application and with the import of the raw temporal data.

Zusammenfassung

Diese Bachelorarbeit entwickelt die up-to-date summary, einen automatischen Mechanismus für die Aggregation an Chronologien von Nährstoffmessungen in der Swiss Feed Database. Da Messungen selten und in unregelmässigen Zeitabständen gesammelt werden, ist eine simple Aggregation der gesamten Chronologie kein repräsentatives Abbild des tatsächlichen Zustands. Wir gehen dieses Problem an, indem wir Trends in der Chronologie der Messungen mit Hilfe einer Reihe von Funktionen aufzeigen, welche die Daten fitten: uniforme fitting-Funktion, lineare und kernel Regressionen. Die experimentelle Evaluation beweist die Skalierbarkeit unseres Ansatzes, Messungen von guter und schlechter Datenqualität zu aggregieren. Des Weiteren trägt diese Bachelorarbeit zur Entwicklung der Feed Database bei, einerseits durch die Einbindung von up-to-date summaries in die Web-Applikation, andererseits durch den Import von rohen temporalen Daten.

Contents

1 Introduction

Aggregates of nutrient measurements are crucial parameters that are necessary for the evaluation and comparison of different feed types by farmers, companies and research institutions. The main problem of computing the aggregates is the poor quality of the available data. Data is very often irregularly taken, sparse or both of them. Thus, aggregates over the entire history of nutrient measurements often lead to imprecisions, i.e., the aggregates do not correctly describe the current quality of a feed.

In this thesis we develop a data driven approach, termed the up-to-date summary, to automatically aggregate the nutrient measurements based on the data distribution. The up-to-date summary detects trends and adjusts the aggregates according to these trends. On a technical level we detect trends in two steps. First, we fit the data with three functions - namely uniform, linear regression and kernel regression. Since the shape of these fitting functions is different, we are capable to distinguish between different trends of the data. Second, we choose such a fitting function that generates the smallest error.

We experimentally evaluate our approach on real world and synthetic data. The results reveal a strong disadvantage of simple aggregation over the entire history. First, in case the data quality is good and there are trends in the data, linear or kernel regressions always result in the smallest error. Second, in case the data quality is poor and independent from the presence of trends, the simple aggregation always results in the highest error. Additionally we show that our approach always recognizes linear and logarithmic trends.

This thesis is done as a part of the Swiss Feed Database and is considered to be an important step towards the temporal Feed Database, that stores the history of nutrient measurements. Besides computation of up-to-date summaries, this thesis makes the following contributions to the temporal Feed Database. First, in collaboration with researchers of Agroscope and University of Zurich, we collected and imported the raw data into the temporal database. Second, we integrated up-to-date summaries into web-application. For that, we developed a user interface which displays the up-to-date summaries with a text and graphically.

This thesis is organized as following. Section 2 presents the current online application of the Feed Database. Sections 3 and 4 describe the temporal Feed Database and the data warehouse. The up-to-date summaries are explained in detail in Section 5. Sections 6 and 7 report the implementation of the online application and the experimental evaluation of the algorithm used to compute the up-to-date summaries. The data import application is described in Section 8. Section 9 concludes the thesis and offers the future work.

2 The Online Feed Database

The Swiss Feed Database is a public service for companies, private farmers and research institutions. The feed data characterizes the quality of grown, imported and synthetic feed types and is derived from feed samples collected from all parts of Switzerland. The project has been started in 2005 by the Swiss Agronomic Institute (Agroscope [3]) in collaboration with the ETH Zurich and has later turned into a collaboration between Agroscope and the University of Zurich.

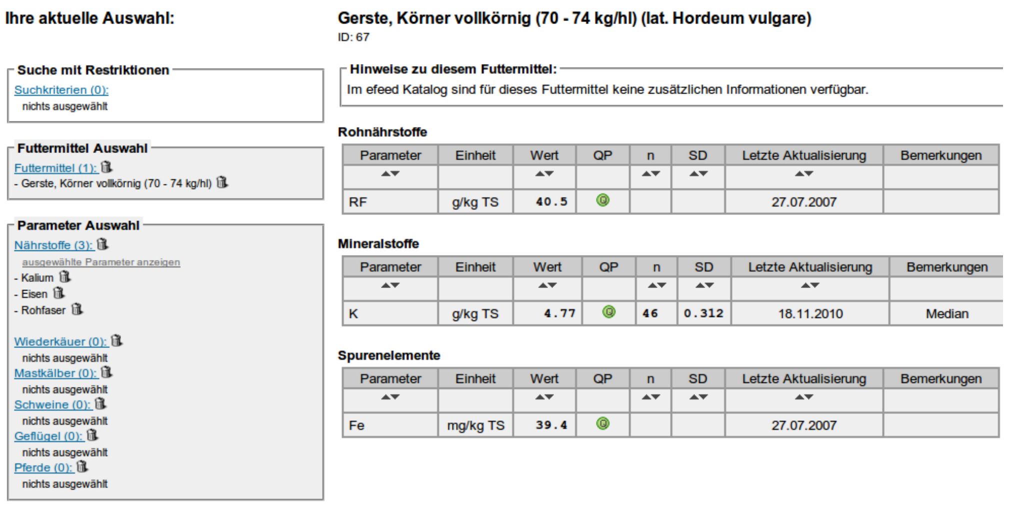
The essence of feed data are measurements of nutrients, that are derived from field samples through chemical analyses. At the moment the Swiss Feed Database stores aggregated nutrient measurements for more than 600 feed types and 155 nutrients. The most recent measurements characterize the current quality of an animal feed. Typically, the selected measurements of the recent time period are aggregated and stored, respectively updated, into the database. For instance, if an aggregated measurement for a certain couple of feed and nutrient already exists in the database, then the old aggregated measurement is overwritten by the new one. The aggregated measurements are used by companies, private farmers and research institutions. The aggregated measurements are critical for planning healthy, effective and cheap animal feed.

Interface:

An online interface gives the users of the database the possibility to search for aggregated measurements. In order to search for an aggregated measurement, at least one feed type and one nutrient should be chosen from the whole list of feed and nutrient types. Users also have the possibility to search for feed types, which fulfill one or more restrictions about their content of the given nutrients. After selecting the desired feeds and nutrients, the aggregated measurements for the given selection are shown in a table. Like mentioned above, the aggregated measurements, which are retrieved here, are nothing else than precomputed aggregations from the most recent measurements.

Example:

Figure 2.1 shows how the aggregated measurements are presented in the web interface of the database. In this case the selected feed is barley and the selected nutrients are potassium, iron and raw-fiber, which are shown on the left side of the figure. Under “Futtermittel” we have the selected feeds and under “Nährstoffe” the nutrients.



Ihre aktuelle Auswahl:

Suche mit Restriktionen

Suchkriterien (0):
nichts ausgewählt

Futtermittel Auswahl

Futtermittel (1):
- Gerste, Körner vollkörnig (70 - 74 kg/hl)

Parameter Auswahl

Nährstoffe (3):
ausgewählte Parameter anzeigen
- Kalium
- Eisen
- Rohfaser

Wiederkäuer (0):
nichts ausgewählt
Mastkälber (0):
nichts ausgewählt
Schweine (0):
nichts ausgewählt
Geflügel (0):
nichts ausgewählt
Pferde (0):
nichts ausgewählt

Gerste, Körner vollkörnig (70 - 74 kg/hl) (lat. Hordeum vulgare)
ID: 67

Hinweise zu diesem Futtermittel:
Im efeed Katalog sind für dieses Futtermittel keine zusätzlichen Informationen verfügbar.

Rohnährstoffe

Parameter	Einheit	Wert	QP	n	SD	Letzte Aktualisierung	Bemerkungen
▲▼		▲▼		▲▼	▲▼	▲▼	
RF	g/kg TS	40.5	●			27.07.2007	

Mineralstoffe

Parameter	Einheit	Wert	QP	n	SD	Letzte Aktualisierung	Bemerkungen
▲▼		▲▼		▲▼	▲▼	▲▼	
K	g/kg TS	4.77	●	46	0.312	18.11.2010	Median

Spurenelemente

Parameter	Einheit	Wert	QP	n	SD	Letzte Aktualisierung	Bemerkungen
▲▼		▲▼		▲▼	▲▼	▲▼	
Fe	mg/kg TS	39.4	●			27.07.2007	

Figure 2.1: Aggregated measurements for barley and nutrients: iron, fiber and potassium

For every nutrient a table displays the aggregated measurement. The first column reports the nutrient name, the second the unity, the third the aggregated measurement for the given nutrient. When clicking on the fourth column more information about the aggregated measurement is displayed, the fifth column reports the number of measurements used to compute the aggregated measurement, the column "SD" states the standard deviation, the last but one column shows the date of the last update and the last column contains possible remarks.

The measurements are aggregated manually by the researchers of Agroscope. There are no fixed rules about how to aggregate measurements, but depending on the quality and quantity of the data, measurements are aggregated differently. For instance, if there exists a big quantity of recent measurements, then a short period of time is chosen to aggregate measurements. Another example is, when a measurement is exceedingly big or small in comparison to the average, then it will be discarded as an outlier and not used for the computation of the aggregated measurement. The time period for the aggregation may also be adjusted based on the feed type. For example, while aggregating measurements of grass, it is more desirable to use short time periods of about three months. In other cases, while aggregating measurements of wheat, long time periods of several years are preferable.

Currently, the Feed Database is extensively extended with temporal and detailed categorical information. Therefore, measurements must be aggregated online with as little input from the user as possible. In the next section we describe in details temporal feed data.

3 The Temporal Feed Data

This section describes the data of the Feed Database.

- Feed: a Feed describes all the animal feeds and some animal's side products such as urine and excrements. A Feed has a name which can be stored in different languages (English, German and French) and an abbreviation of its name. Each Feed is uniquely identified by an artificial ID. Example:

id	name_e	name_d	name_f	abbreviation
1	Barley	Gerste	Orge	GER
2	-	Erdnüssöl	-	ERDO

- Nutrient: a Nutrient has a name which can be stored in different languages (English, German and French) and an abbreviation of its name. Every Nutrient is uniquely identified by an artificial ID. A Nutrient can be a nutrient matter or a nutritional value for a given species of animal. Example:

id	name_e	name_d	name_f	abbreviation
1	Lysine	Lysin	Lysine	LYS
2	-	Ileal verdauliches lysin schwein	Lysine digestible iléale porc	IVLS

- Sample: a Sample of a given feed stores its origin, its harvest date, the sampling date, the arrival date at the laboratory and the analysis date. Almost every Sample has a LIMS-number, which uniquely identifies a Sample, and an additional artificial ID is needed to always identify a Sample uniquely. On each Sample one or more measurements can be done. Example:

id	LIMS_nr	origin	harvest_date	sampling_date	arrival_date	analysis_date
1	332434-3	Bern	19.03.2004	08.2004	-	-
2	947283-1	6596	2007	-	-	13.05.2007

- Measurement: a Measurement stores the chemical analysis value of a certain Nutrient in a particular Sample. A Measurement is uniquely identified by an artificial ID. Example:

id	value	sample	nutrient
1	8.30432	2	1
2	10.438	1	2

- Species: a Species of domestic animals has a name which can be stored in different languages (English, German and French) and is uniquely identified by an artificial ID. Example:

id	**name_e**	**name_d**	**name_f**
1	Pig	Schwein	Porc
2	Horse	Pferd	-

- NutrientGroup: a NutrientGroup categorizes the nutrients according to their biological properties. For example we can find groups of minerals, carbohydrates or vitamins. A NutrientGroup contains one or more Nutrients, has a name which can be stored in different languages (English, German and French) and is uniquely identified by an artificial ID. A NutrientGroup may have a parent NutrientGroup and also be the parent of one or many other NutrientGroups. A NutrientGroup can also be bound to an animal Species. Groups which are bound to an animal Species contain rates of digestibility of various Nutrients for the given Species. Example:

id	**name_e**	**name_d**	**name_f**	**parent_group**	**animal_species**
1	-	Mineralstoffe	Minéraux	Nährstoffe	-
2	-	Schweine	Porcs	Nährwerte	Schweine

- FeedGroup: a FeedGroup groups different categories of Feeds. A FeedGroup contains one or more Feeds, has a name which can be stored in different languages (English, German and French) and is uniquely identified by an artificial ID. A FeedGroup may have a parent FeedGroup and also be the parent of one or many other FeedGroups. Example:

id	**name_e**	**name_d**	**name_f**
1	-	Stroh	Pailles
2	-	Getreidekörner	Grains de céréales

The feed data exhibits the following properties. First, it is sparse in all types of attributes: temporal, categorical and spacial. For example, it is usual that the harvest and analysis dates are given but the sampling and arrival date are missing. Also the LIMS-number and an origin are not always given. Second, granularity of temporal and spacial attributes are not fixed. For example, sometimes a city name or a canton in Switzerland is given and sometimes a postal code. The dates are entered entirely (e.g. 04.10.2003) or just as a year (e.g. 2007).

4 Data Warehouse

This section describes the data warehouse that stores temporal feed data. The data warehouse is a non normalized database, that in general decreases the querying time. That is achieved by building a fact table, where all the numerical values, which are going to be used during the analysis, are stored. Then, other tables, called dimensions, are built to give a context to the values of the fact table, where the dimensions together are assumed to uniquely determine the measurement [5].

For instance, "Fact Table" is the fact table of our database, in which all the measurement values and the foreign keys referencing to its six dimensions are stored. The dimensions store information about the Sample, the Animal Species, the Feed, the Origin, the Nutrient and the Time for every measurement. The LIMS-number is also reported in the "Fact Table" for every measurement in order to not have to make a join with the "Sample" table every time a LIMS-number is required. In this database every measurement is stored as a single value and never as an aggregated value. Figure 4.1 shows the diagram of the database.

In addition to the data described in the section above, the data warehouse also uses two separate tables to describe the origin of a sample as well as its harvest, sampling, arrival and analysis date.

- Origin: this table stores the origin of a Sample. An Origin is composed of a country, a canton or region, a city, an altitude, a postal code and the animal density. In case the sample comes from an external agency the table additionally provides a field for the agency name, the agency address, an e-mail address and a web page. Example:

country	**region**	**city**	**altitude**	**ZIP**	**animal_density**	**a_name**	**a_address**	**e-mail**	**website**
CH	Waadt	Aigle	415	1860	-	-	-	-	-
DE	Bayern	Hof	315	95032	-	F&B	Gersteweg 1	-	-

- Time: this table stores the exact harvesting date of a sample, the sampling date, the date of arrival at the laboratories and the analysis date. Example:

harvest_date	**sample_date**	**arrival_date**	**analysis_date**
20.08.2005	21.08.2005	23.08.2005	25.08.2005
03.02.2007	-	-	-

Figure 4.2 illustrates a reduced example of the data stored in the data warehouse:

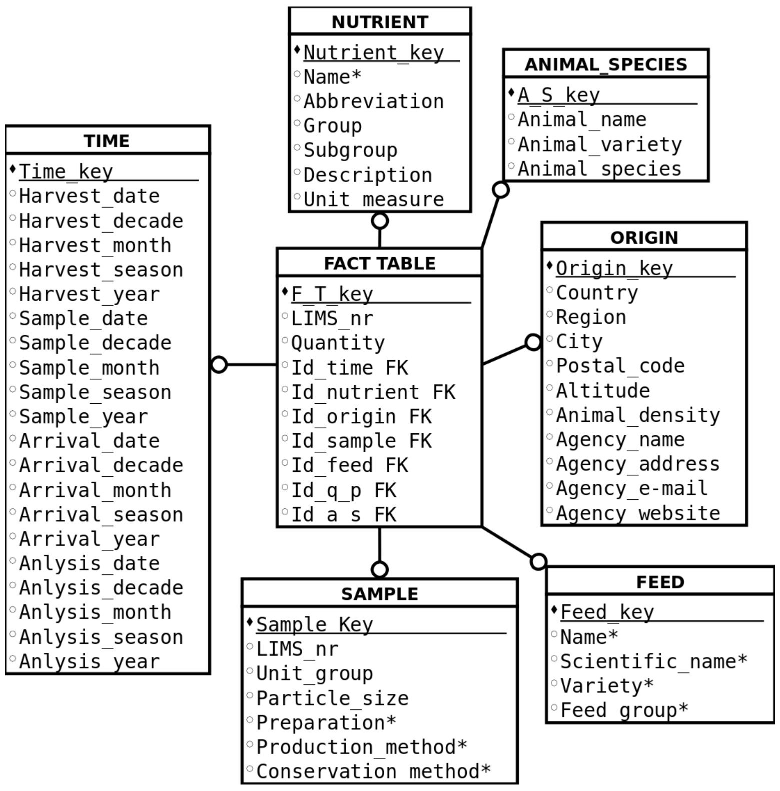
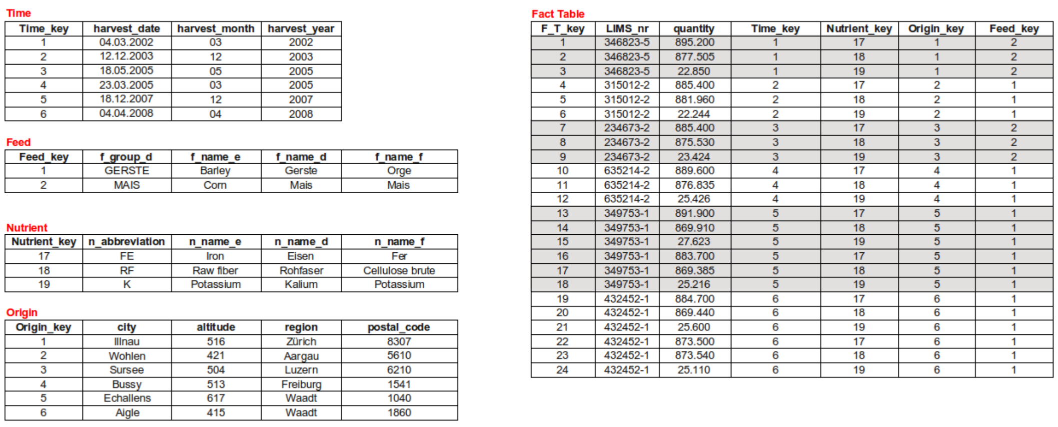


Figure 4.1: Data Warehouse



Time

Time_key	harvest_date	harvest_month	harvest_year
1	04.03.2002	03	2002
2	12.12.2003	12	2003
3	18.05.2005	05	2005
4	23.03.2005	03	2005
5	18.12.2007	12	2007
6	04.04.2008	04	2008

Feed

Feed_key	f_group_d	f_name_e	f_name_d	f_name_f
1	GERSTE	Barley	Gerste	Orge
2	MAIS	Corn	Mais	Mais

Nutrient

Nutrient_key	n_abbreviation	n_name_e	n_name_d	n_name_f
17	FE	Iron	Eisen	Fer
18	RF	Raw fiber	Rohfaser	Cellulose brute
19	K	Potassium	Kalium	Potassium

Origin

Origin_key	city	altitude	region	postal_code
1	Illnau	516	Zürich	8307
2	Wohlen	421	Aargau	5610
3	Sursee	504	Luzern	6210
4	Bussy	513	Freiburg	1541
5	Echallens	617	Waadt	1040
6	Aigle	415	Waadt	1860

Fact Table

F_T_key	LIMS_nr	quantity	Time_key	Nutrient_key	Origin_key	Feed_key
1	346823-5	895.200	1	17	1	2
2	346823-5	877.505	1	18	1	2
3	346823-5	22.850	1	19	1	2
4	315012-2	885.400	2	17	2	1
5	315012-2	881.960	2	18	2	1
6	315012-2	22.244	2	19	2	1
7	234673-2	885.400	3	17	3	2
8	234673-2	875.530	3	18	3	2
9	234673-2	23.424	3	19	3	2
10	635214-2	889.600	4	17	4	1
11	635214-2	876.835	4	18	4	1
12	635214-2	25.426	4	19	4	1
13	349753-1	891.900	5	17	5	1
14	349753-1	869.910	5	18	5	1
15	349753-1	27.623	5	19	5	1
16	349753-1	883.700	5	17	5	1
17	349753-1	869.385	5	18	5	1
18	349753-1	25.216	5	19	5	1
19	432452-1	884.700	6	17	6	1
20	432452-1	869.440	6	18	6	1
21	432452-1	25.600	6	19	6	1
22	432452-1	873.500	6	17	6	1
23	432452-1	873.540	6	18	6	1
24	432452-1	25.110	6	19	6	1

Figure 4.2: Example of data stored in the database

In this example we stored the measurements done on six different samples which come from six different places in Switzerland. In the “Fact Table” the measurements of the different samples are divided by gray and white coloration. Every sample is from a given feed - in this case we have two different kinds of feeds (barley and corn). There are 24 measurements which report the values for three different nutrients (iron, fiber and potassium) analyzed from each sample.

5 Up-To-Date Summaries

There are four aggregates, *min*, *max*, *mean* and *var*, which must be computed from a history of nutrient measurements. *min* and *max* are the minimal and maximal values of nutrient measurements, and their computation does not depend on the data distribution or data quality. In an ideal case, when there is a large number of nutrient measurements and their values are distributed uniformly, the *mean* and the *var* are the average and the variance of the most recent measurements in the past.

The *up-to-date summary* is composed by aggregated *mean* and *var* values that are computed based on the trends of the data, and, in case of poor data quality, describe the real world state more accurately than simple aggregation .

5.1 Impact of Data Quality

The up-to-date summary should report the real world state in the most possible accurate way. This can be ensured if the quality of the data is good, i.e., if there are many measurements which are taken periodically. Figure 5.1 shows how good quality data looks like. We have many measurements taken regularly over time. Therefore, the up-to-date summary can be computed by simply averaging all measurements, yielding 221 as the output.

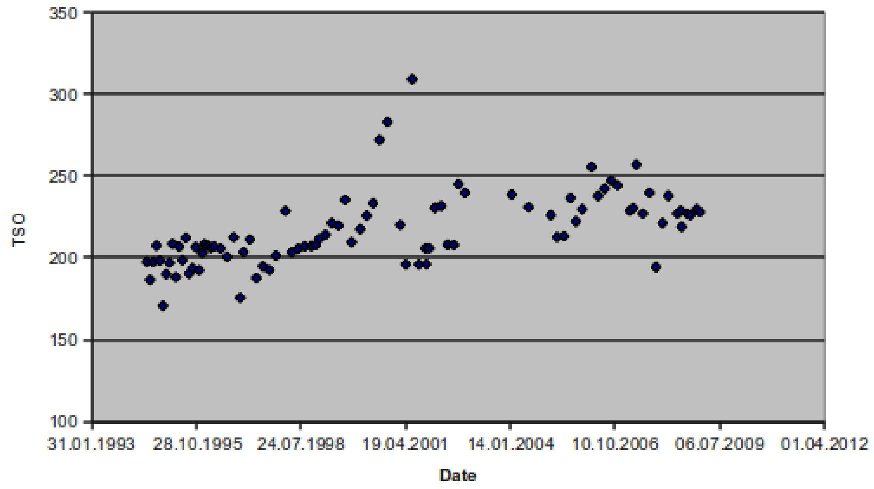


Figure 5.1: Good quality data, measurements of “TSO” in “Biertreber”

The key challenge in computing the up-to-date summary is that the quality of the data is poor. It is always the case that measurements are not taken regularly and are sparse in time.

Figure 5.2 illustrates the case where the number of measurements is sufficient, but the measurements are not taken regularly. There are three regions where the measurements are concentrated and in each region the number of measurements is different: the first region between 1994 and 1996 contains the most measurements, while in the second region between the years 2001 and 2002 there are only eight. Averaging the measurements of all the three regions yields value 162 that is mostly influenced by the measurements of the first region, i.e., by the past history. However, if we take the average measurements of the last region only, we come to a value of 158, which is more representative of the present state of the real world.

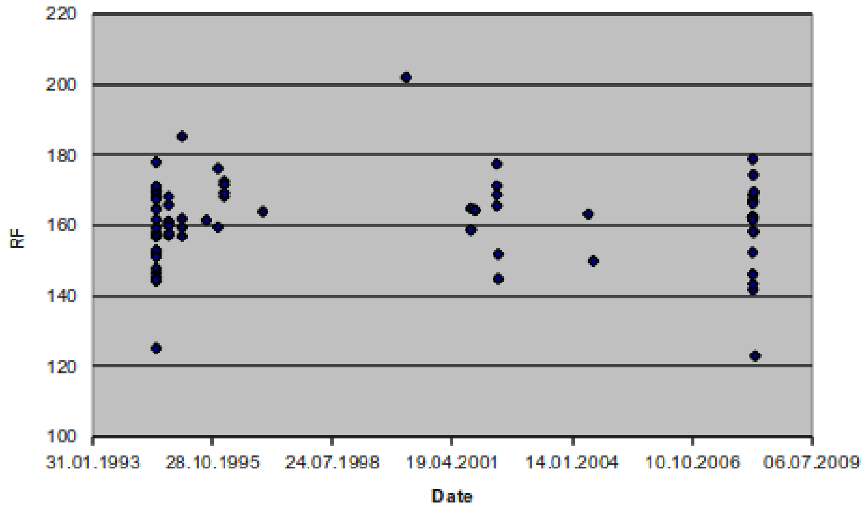


Figure 5.2: Irregularly taken data, measurements of “RF” in “Biertreber”

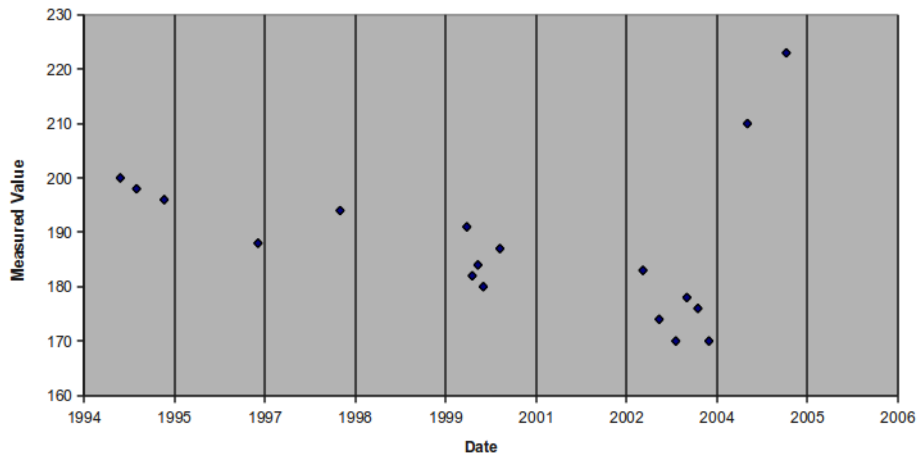


Figure 5.3: Sparse data

Figure 5.3 illustrates the case where data is collected regularly but is sparse. In this example, averaging over the entire history and over the most recent measurements yields substantially different results. The average for the last two years 2004 and 2005 is 216.5 and the average over the entire history is 188.2. However, we can observe the linear trend of the data, i.e., the value of the measurements decreases linearly, and thus both averages are unrepresentative of the current state of the real world.

5.2 Data Driven Approach

Our approach to compute the up-to-date summaries is data driven. For each history of nutrient measurements we recognize a possible trend in the data distribution by fitting it with one of the pre-configured function. Then the value of the function at the most recent date is the average for the up-to-date summary and the root mean squared error of fitting the data is the variance for the up-to-date summary.

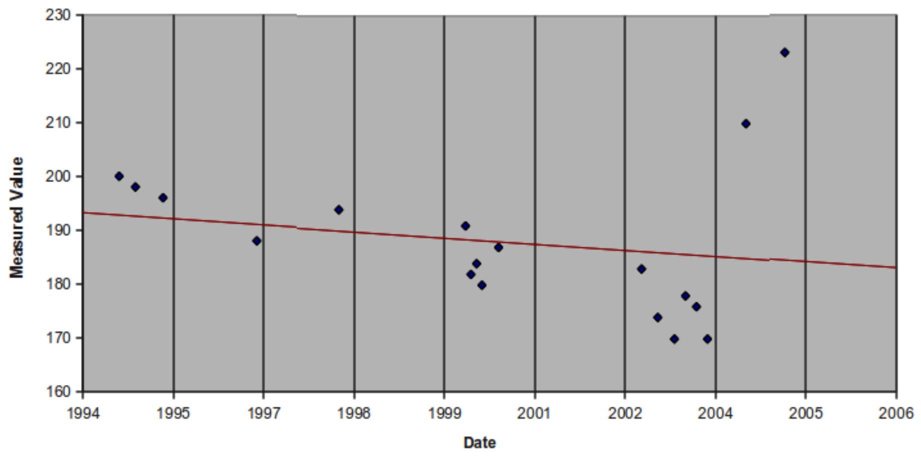


Figure 5.4: Sparse data with fitting line

Figure 5.4 illustrates our approach on the sparse data. The line in the figure represents linear regression that fits the data with mean squared error of 10.3. The average for the up-to-date summary is the value of the regression at 2005, i.e., 185, and 10.3 is the variance for the up-to-date summary.

In our approach we distinguish between three fitting functions. The uniform function is a line that goes through the average value over the entire history and is parallel to the x-axis. The uniform function best describes the data with no trends. Linear regression is used to recognize linearly increasing and decreasing trends. The kernel regression applies for the case when the trend in the data is not linear. The whole process of computing the up-to-date summary consists of three steps:

1. Fitting the data with all three functions;

2. Selecting the fitting function;

3. Computing the average and the variance from the selected fitting function.

The best fitting function is chosen based on the mean squared error (MSE), i.e., we compute the MSE for each of the three functions and select the one which generates the smallest error.

We use M to denote a set of measurements and (t_i, m_i) to denote an element of M. t_i is the timestamp as an integer and m_i is the value of a measurement. For example in Figure 5.4, the set of measurements M corresponds to the crowd of all blue squares. For each square the coordinate on the x-axes is time t_i and the coordinate on the y-axis is value m_i. Table 5.1 summarizes the notation used in this section.

Notation	**Meaning**	**Example**
M	Set of measurements	$\{(1220870000, 1.23), (1220870000, 2.312)\}$
$(t_i, m_i) \in M$	Measurement in the set M	$(8003840000, 10.122)$
$t_i \in \mathbb{Z}$	Timestamp as an integer representing a date	2011.June.10 correspond to the timestamp 1317852000
$m_i \in \mathbb{R}$	Measurement value	23.231
n	Total number of measurements in the set M	-
i	Index of a measurement	-
f	Fitting function	-
$f(t)$	Value of the fitting function f at timestamp t	-
$MSE(M, f)$	Mean squared error for the set M and function f	-

Table 5.1: Notation

The mean squared error is defined as following. Given a set of measurements $M = \{(t_1, m_1), ..., (t_n, m_n)\}$ and a fitting function f, the $MSE(M, f)$ is the average of the squared differences between the given value m_i at t_i and the value of f at t_i:

$$MSE(M, f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(t_i))^2 \tag{5.1}$$

5.3 Uniform Fitting Function

Definition: let $M = \{(t_1, m_1), ... (t_i, m_i), ... (t_n, m_n)\}$ be a set of measurements. The uniform fitting function f is:

$$f(t) = \frac{1}{n}\sum_{i=1}^{n} m_i \tag{5.2}$$

Explanation: At each timestamp t_i, the uniform fitting function has the same value. That is equal to the average of all measurements.

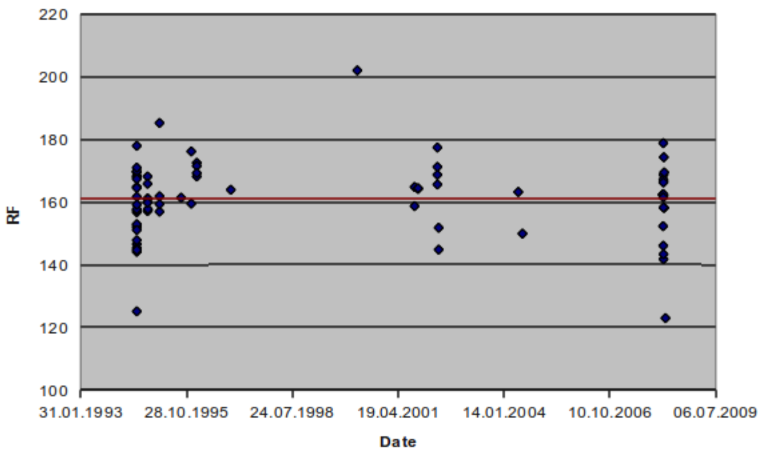


Figure 5.5: Uniform fitting function

Example: Figure 5.5 illustrates fitting the data with uniform fitting function.

5.4 Linear Regression

For the linear regression we adopt the description in [8] as following:

Definition: let $M = \{(t_1, m_1), \dots (t_i, m_i), \dots (t_n, m_n)\}$ be a set of measurements. The linear regression f is:

$$f(t) = a + b \cdot t \quad ; a, b \,\epsilon\, \mathbb{R} \tag{5.3}$$

where: $b = \frac{n \cdot \sum_{i=1}^{n} m_i \cdot t_i - \sum_{i=1}^{n} t_i \cdot \sum_{i=1}^{n} m_i}{n \cdot \sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2}$ $\quad ; a = \frac{\sum_{i=1}^{n} m_i - b \cdot \sum_{i=1}^{n} t_i}{n}$

Explanation: Linear regression fits the data with a line that has parameters a and b. a is the intersection point on the vertical axis and b is the slope of the line. a and b are computed with least-squares approach, that minimizes the mean squared error [8].

Example: Figure 5.6 illustrates a set of measurements, that exhibit an increasing linear trend. The linear regression is $f(t) = 0.0074t - 56.9169$.

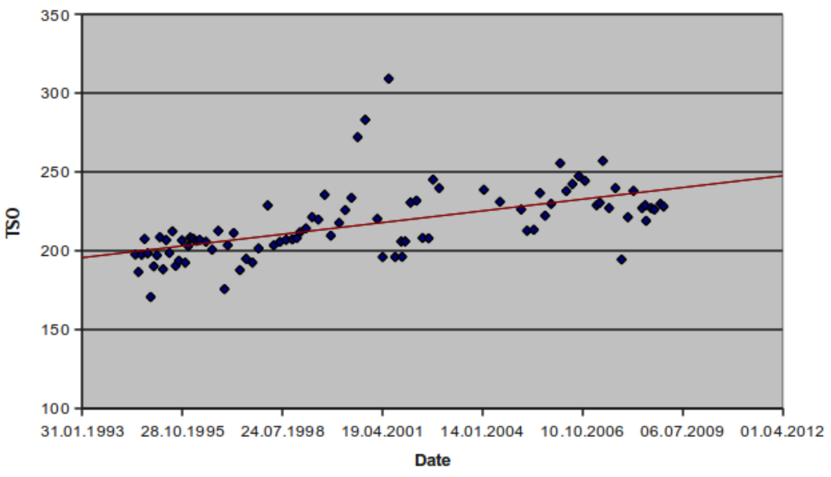


Figure 5.6: Linear regression

5.5 Kernel Regression

We use the kernel regression method to detect non-linear trends in the data. The kernel regression method builds a fitting function $f(t)$, with the help of a non-parametric density estimator. That is an advantage, since no assumptions are made about the data distribution. For instance, if we want to estimate the density of the data using a normal distribution $N(\mu, \sigma)$, we would optimize the parameters μ (the mean) and σ (the standard deviation), and have the same shape of the density distribution for any dataset. The kernel density estimator relies on the kernel function K_h of width h. First, the kernel function K_h is placed at each data point and next, at each point of the domain, where overlapping kernel functions are summed up [11].

Example: Consider Figure 5.7a, kernel functions are denoted by white bells. We place a kernel function at the timestamp of each nutrient measurement. Summing up all the kernel functions yields the density function that is represented by a light gray color in the figure.

The width of the kernel function h is the most important parameter for accurate density estimation. Small h values make the density function sensitive to small changes in the data distribution, while high h values make the density function smooth. It is possible to compute the optimal h value based on the standard deviation of the data.

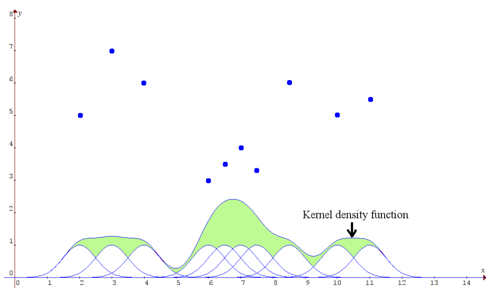
We adopt the definitions of kernel regression in [10, 11] as following:

Definition: let $M = \{(t_1, m_1), \ldots (t_i, m_i), \ldots (t_n, m_n)\}$ be a set of measurements. The kernel regression f is :

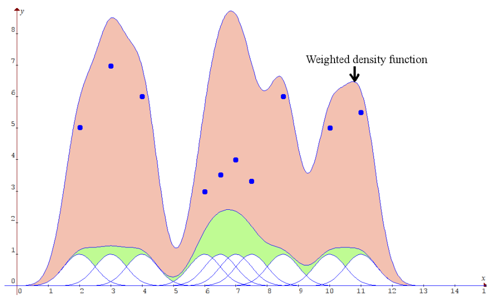
$$f(t) = \frac{\sum_{i=1}^{n} m_i \cdot K_h(t, t_i)}{\sum_{i=1}^{n} K_h(t, t_i)} \tag{5.4}$$

where:

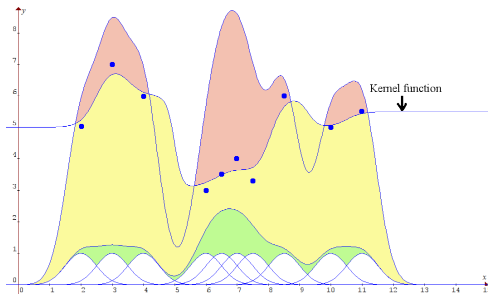
- $K_h(t,t_i) = exp\left(-\frac{(t-t_i)^2}{2\cdot h^2}\right)$ *is the Gaussian kernel function*
- $h = \left(\frac{4\cdot\sigma_t^5}{3\cdot n}\right)^{\frac{1}{5}}$ *is the optimal bandwidth*
- $\sigma_t = \sqrt{\sum_{i=1}^{n}\left(t_i - avg\left(t\right)\right)^2}$ *is the standard deviation of the timestamps*
- $avg\left(t\right) = \frac{1}{n}\sum_{i=1}^{n} t_i$ *is the average of the timestamps*



(a) Non-weighted density function



(b) Weighted density function



(c) Kernel regression function

Figure 5.7: Computation of the kernel regression

Explanation: Computation of kernel regression can be viewed as the outcome of three steps. During the first step we compute the density function of timestamps that are present in the set of measurements M. That step corresponds to the denominator in Equation 5.4. In the second step we compute the weighted density function of the timestamps, where the weights are nutrient measurements m_i. That corresponds to the numerator in Equation 5.4. In the last step we divide weighted with non-weighted density functions.

Example: Consider Figure 5.7 and the set of measurements $M = \{(2,5), (3,7), (4,6), (6,3), (6.5,3.5), (7,4), (7.5,3.3), (8.5,6), (10,5), (11,5.5)\}$. Figure 5.7a illustrates the denominator of Equation 5.4, i.e., the density function of timestamps. Figure 5.7b illustrates the numerator of Equation 5.4, i.e., the weighted density function of timestamps. The curve in Figure 5.7c is the kernel regression achieved by dividing weighted with non-weighted density functions.

5.6 Example of Computation of an Up-To-Date Summary

let M be a set of 24 measurements:

M= {(02.10.2001, 34.19), (02.10.2001, 34.03), (02.10.2001, 39.67), (30.04.2002, 40.52), (30.04.2002, 44.53), (30.04.2002, 42.73), (30.01.2009, 39.06), (18.02.2009, 43.46), (18.02.2009, 42.16), (18.02.2009, 42.31), (18.02.2009, 43.53), (13.03.2009, 36.61), (13.03.2009, 36.55), (04.06.2009, 80.49), (04.06.2009, 36.76), (23.07.2009, 44.14), (13.08.2009, 40.44), (05.11.2009, 39.91), (05.11.2009, 39.60), (11.11.2009, 43.72), (27.11.2009, 44.45), (30.11.2009, 39.83), (21.06.2010, 44.05), (21.06.2010, 44.14)}

Table 5.2 reports the MSE for the three fitting functions.

Method	MSE
Uniform	73.1794
Linear	69.6847
Kernel	69.9117

Table 5.2: MSEs of the three fitting functions

Linear regression is the function which returns the smallest MSE, therefore it is chosen to compute the up-to-date summary.

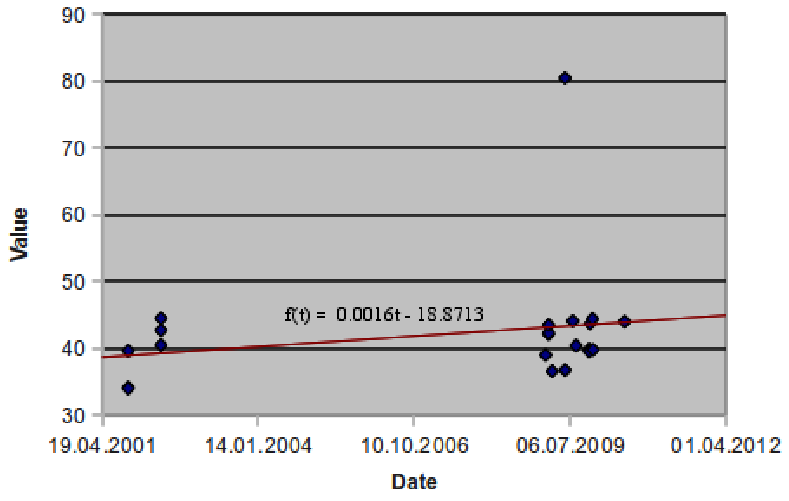


Figure 5.8: Graphical visualization of the dataset M and the linear regression

The *mean* value for the up-to-date summary is the value of the linear regression at the most recent date, in this case the *mean* is equal to 43.9501. The *avg* for the up-to-date summary is equal to the root of 69.6847. Table 5.3 reports all the values for the up-to-date summary of the dataset M.

mean	43.9501
var	8.347
min	34.03
max	80.49

Table 5.3: Up-to-date summary for the dataset M

6 Implementation

The implementation is divided into two parts: the online computation of up-to-date summaries and the graphical user interface. The user interface gives the users of the feed database the possibility to set up input parameters, which are used to compute the desired up-to-date summary. The database used at the moment contains data only from a single type of feed (hay), so the user of the online application will not have the possibility to choose between different types of feeds. The computation of summaries is done primarily in PHP and some little parts of javascript, while the graphical visualization of the results is done by a javascript. This application is implemented as an extension of the prototype from Francesco Cafagna.

6.1 Graphical User Interface

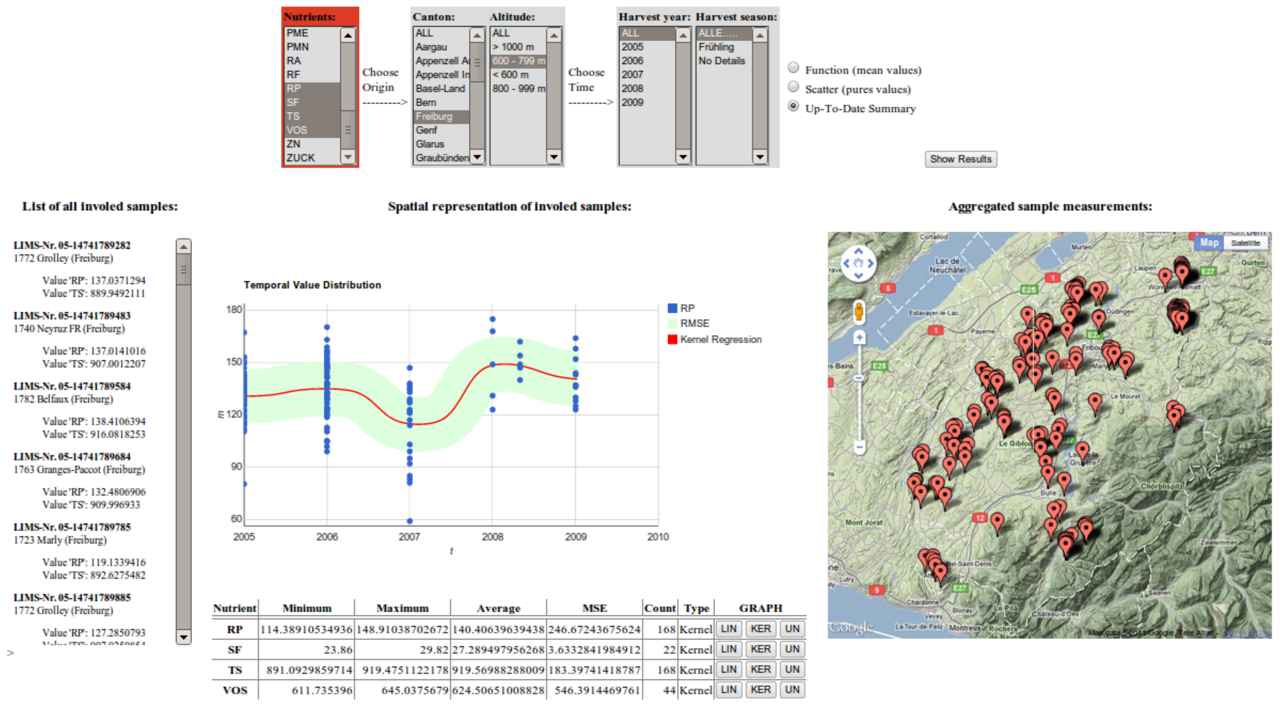
The graphical user interface permits to set up five different parameters:

- A nutrient: the name of a nutrient
- A region: a canton in Switzerland
- An altitude: four different altitude regions
- A harvest year: a choice between all the possible harvest years
- A harvest season: a choice between all the possible harvest seasons

For each setup of input parameters the results are divided into three parts, like shown in Figure 6.1:

On the left column we can find the list of the samples involved in the computation of the summary. On the right side we have the representation of the geographical distribution of the samples. Finally, in the middle of the page, a table reports the up-to-date summaries for all the selected nutrients. The last-but-one column of the table specifies the fitting function type used to compute the summary. However, for each nutrient it is possible to generate three different graphs showing how the measurements are fitted using the three different fitting functions described in the previous section.

Selecting all the measurements for the nutrient “Phosphor” taken in canton “Zug” returns the results shown in Figure 6.2. All the measurements are illustrated using blue points, on the x-axis we can find the harvest date of the samples and on the y-axis the value for each measurement. The measurements have been fitted using a uniform, a linear and a kernel function. The red



List of all involed samples:

LIMS-Nr. 05-14741789282
1772 Grolley (Freiburg)

Value 'RP': 137.0371294
Value 'TS': 889.9492111

LIMS-Nr. 05-14741789483
1740 Neyruz FR (Freiburg)

Value 'RP': 137.0141016
Value 'TS': 907.0012207

LIMS-Nr. 05-14741789584
1782 Belfaux (Freiburg)

Value 'RP': 138.4106394
Value 'TS': 916.0818253

LIMS-Nr. 05-14741789684
1763 Granges-Paccot (Freiburg)

Value 'RP': 132.4806906
Value 'TS': 909.996933

LIMS-Nr. 05-14741789785
1723 Marly (Freiburg)

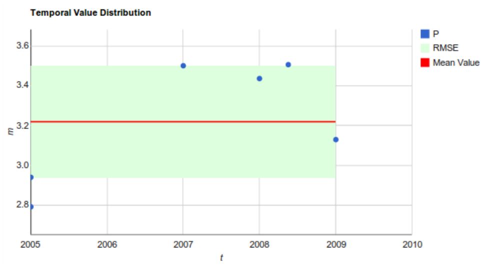
Value 'RP': 119.1339416
Value 'TS': 892.6275482

LIMS-Nr. 05-14741789885
1772 Grolley (Freiburg)

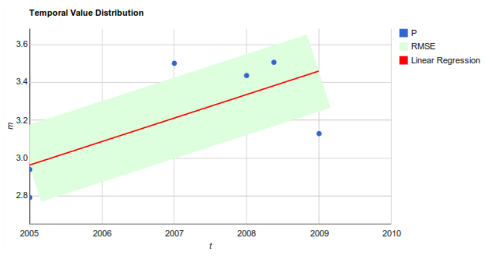
Value 'RP': 127.2850793

Nutrient	Minimum	Maximum	Average	MSE	Count	Type	GRAPH		
RP	114.38910534936	148.91038702672	140.40639639438	246.67243675624	168	Kernel	LIN	KER	UN
SF	23.86	29.82	27.289497956268	3.6332841984912	22	Kernel	LIN	KER	UN
TS	891.0929859714	919.4751122178	919.56988288009	183.39741418787	168	Kernel	LIN	KER	UN
VOS	611.735396	645.0375679	624.50651008828	546.3914469761	44	Kernel	LIN	KER	UN

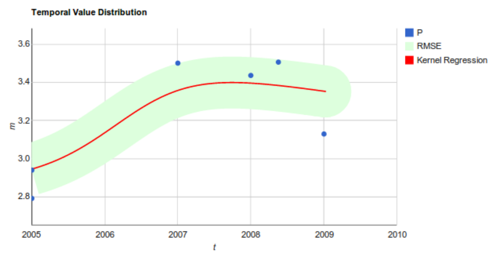
Figure 6.1: Online application interface



(a) Uniform function, RMSE = 0.2817



(b) Linear regression, RMSE = 0.2032



(c) Kernel regression, RMSE = 0.1234

Figure 6.2: Fitting the data with different functions

min	*max*	*mean*	*var*	Function type
2.7917	3.5053	3.2169	0.2798	Uniform
2.7917	3.5053	3.4572	0.2029	Linear
2.7917	3.5053	3.3440	0.1232	Kernel

Table 6.1: Up-to-date summaries for nutrient “Phosphor” in canton “Zug”

lines in the graphs represent the functions and their root mean squared error (RMSE) is displayed in light gray. The width of the light gray area indicates the amplitude of the RMSE. In Figure 6.2, the best fitting function is the kernel one, which has the smallest RMSE, followed by the linear regression function. The uniform function (or mean value line) has the highest RMSE and performs rather badly in comparison to the other two.

The up-to-date summaries generated using the three different fitting functions are diagrammed in Table 6.1.

The average value is calculated by taking the date of the last measurement, in this case the date in t_6, and by calculating the value of $f(t_6)$ for the given fitting function. This example is illustrated in Figure 6.3, the average value is reported on the y-axis.

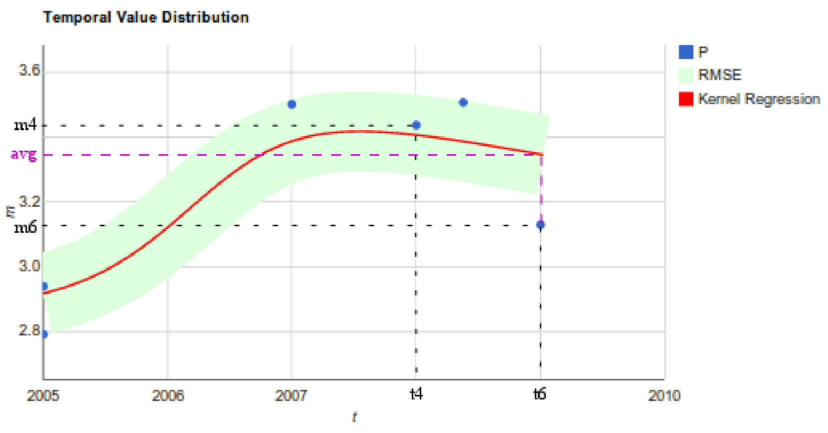


Figure 6.3: Average calculation

6.2 Computation of Up-To-Date Summaries

The computation of up-to-date summaries can be divided into three steps. As a first step a query containing the selected parameters - described in the section above - is generated. As a second step all the measurements selected from the query are retrieved from the database. For every selected nutrient an up-to-date summary is computed.

Algorithm 6.1 generates the up-to-date summary for a given set of measurements M, having a total number of measurements n. The intermediate results $sumM$, $sumT$, $prodMT$, $prodTT$ are used to calculate the linear and the uniform fitting functions, the intermediate result $mseT$ is used in the kernel regression formula. Using the results of a first scan (lines 6-11) of the measurements, the mean value of all measurements and the parameters for the linear regression (intercept and slope) are calculated (lines 12-15), with a second scan of the data (lines 16-18) the optimal bandwidth for the Gaussian kernel function is calculated.

Algorithm 6.2 calculates the mean squared error (MSE) for all three fitting functions (lines 4-11), compares them and, taking the best fitting function, returns the up-to-date average value like described in the section above (lines 12-16). The data are scanned one more time (lines 4-8) in order to calculate the MSE of the uniform and the linear functions, and for the kernel regression the scan of the measurements becomes quadratic, scanning all the data once more for every value.

Algorithm 6.3 takes a date t as input and returns the measurement value (y-value) of the kernel function for the given date. All the implemented formulas are the ones described in sections about the computation of fitting functions. The whole application is implemented in PHP and javascript and is about 560 lines of code long.

Algorithm 6.1 *summaryGenerator* (M, n)

```
input : a set of measurements $M$, the total number of measurements $n$ in $M$
1.  $meanValue$; //the mean value of all $m_i$
2.  $a$; //intercept of the linear regression line
3.  $b$; //slope of the linear regression line
4.  $bandwidth$; //the optimal bandwidth for Gaussian kernel function
5.  $sumM$, $sumT$, $prodMT$, $prodTT$, $mseT$; //intermediate results
6.  for $i = 1, ..., n$ do
7.          $sumM+ = m_i$;
8.          $sumT+ = t_i$;
9.          $prodMT+ = m_i * t_i$;
10.         $prodTT+ = t_i * t_i$;
11. end for
12. $meanValue = sumM/n$;
13. $b = \frac{(n*prodMT)-(sumT*sumM)}{n*prodTT-sumT^2}$;
14. $a = \frac{sumM-b*sumT}{n}$;
15. $meanT = sumT/n$;
16. for $i = 1, ..., n$ do
17.         $mseT+ = (t_i - meanT)^2$;
18. end for
19. $sdT = \sqrt{mseT/n}$; //Standard deviation of T
20. $bandwidth = \left(\frac{sdT^5*4}{3*n}\right)^{\frac{1}{5}}$
21. return $bestFitting(M, n, meanValue, a, b)$;
```

Algorithm 6.2 *bestFitting* (M, n, $meanValue$, a, b)

input : a set of measurements M, the total number of measurements n in M, $meanValue$ as the mean value of all m_i, the intercept of the linear regression line a, the slope of the linear regression line b

```
 1. $mseUn$; //the mean squared error for the uniform function
 2. $mseLin$; //the mean squared error for the linear function
 3. $mseKer$; //the mean squared error for the kernel function
 4. for $i = 1, ..., n$ do
 5.     $mseUn += (m_i - meanValue)^2$ ;
 6.     $mseLin += (m_i - (a + b * t_i))^2$ ;
 7.     $mseKer += (m_i - kernelFunction\,(t_i, ))^2$ ;
 8. end for
 9. $mseUn = mseUn/n$;
10. $mseLin = mseLin/n$;
11. $mseKer = mseKer/n$;
12. if $mseKer \leq mseLin$ && $mseKer \leq mseUn$ then
13.     return $kernelfunction(t_n)$;
14. else if $mseLin \leq mseKer$ && $mseLin \leq mseUn$ then
15.     return $a + b * t_n$;
16. else
17.     return $meanValue$;
```

Algorithm 6.3 *kernelFunction* (M, n, t)

input : a set of measurements M, the total number of measurements n in M and a date t

```
1. $numerator, denominator$; //of the kernel function
2. for $i = 1, ..., n$ do
3.         $gaussianKernelFunction = exp\left(-\frac{(t-t_i)^2}{2*bandwidth^2}\right)$;
4.         $numerator+ = m_i * gaussiankernelFunction$;
5.         $numerator+ = gaussiankernelFunction$;
6. end for
7. return $numerator/denominator$;
```

6.3 Graphical Visualization of the Results

The javascript method, that is used to draw the graphs, takes a set of measurements, a fitting line or curve and the value of the MSE as input. The graphs are generated using the Google Chart Tool package [1], importing two javascript files [2, 4]. The points to draw into the graph have to be entered into a two-dimensional array, specifying the x-value (the date) and the y-value (the measurement value).

A particularly challenging task has been the graphical representation of the MSE: since the Google Chart Tool package only permits to draw points or points connected by a line, it has been impossible to define an area representing the MSE. This problem has been solved by defining a bright-colored line having a width that is two times bigger than the MSE and passes exactly through the fitting function points. The only problem has been, that - since the MSE squares every error - the resulting MSE line has been too wide (for several errors bigger than 1) or too small (for several errors smaller than 1), resulting in an area covering the whole graph or just the fitting function line. The solution for this last problem has been to take the root of the MSE, thus decreasing the width for big MSEs and increasing the width for the small MSEs. For instance, a MSE = 100 would result in a RMSE = 10 and a MSE = 0.01 in a RSME = 0.1.

7 Experimental Evaluation

In this chapter, we experimentally evaluate our approach to compute the up-to-date summaries. The results show that linear and logarithmic trends are always recognized - independently of how good the data are collected. In addition the results show that the uniform fitting function, i.e. averaging across the entire history, is never chosen, because of the highest MSE. Especially in the cases where the data are collected irregularly, simple averaging leads to very big imprecision estimating the real values.

The first experiment evaluates the performance of the algorithm. Figure 7.1 reports the execution time of the algorithm for different values of n, where n is the total number of measurements. The data used in this experiment come from different selections of nutrients and regions (cantons) of origin. There are three different curves in the graph, each of them reports the computation time for a different fitting function. We made the following observations: while for the uniform and linear function the time elapsed is always some milliseconds scaling linearly with n, the time elapsed using the kernel function increases quadratically with n like shown very well in Figure 7.1. This is due to the definition of kernels. Moreover, object oriented PHP is not optimized to work with arrays, thus the total time of computation for the kernel regression is very high.

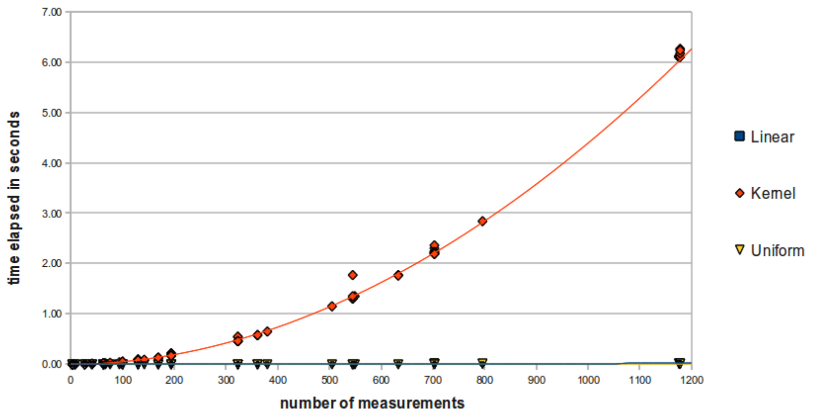


Figure 7.1: Time elapsed for the computation of the up-to-date summary using linear, kernel and uniform functions in correspondence to the number of measurements

The next experiments evaluate the precision of our method. We investigate the following cases:

- Case 1: We verify if the algorithm always detects linear trends. In this experiment the data are assumed to be taken regularly over time, the number of measurements n, the variance of the data and the slope of the line are varying.
- Case 2: The second case extends the first case for irregularly collected data.
- Case 3: The third case verifies if the algorithm always detects logarithmic trends with the help of kernel regression. In this experiment the data is assumed to be taken regularly over time, the number of measurements n, the variance of the data and the curvature of the logarithm are varying.
- Case 4: The last case extends the third case for irregularly collected data.

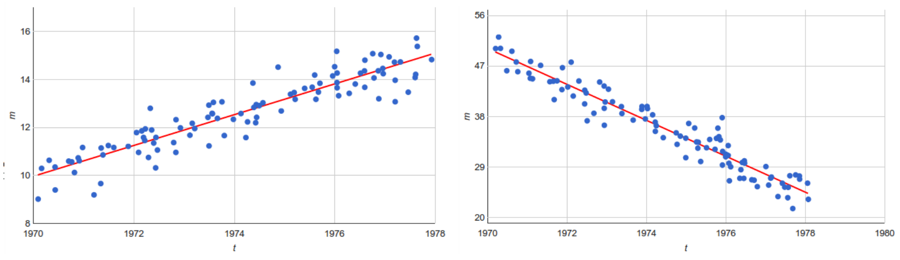
The four cases described have been evaluated using synthetically generated data. For the first two cases the data is generated according to the equation of a line $f(t) = a + b * t$, where a and b are real numbers. For the last two cases the data is generated according to the equation of the logarithmic curve $f(t) = log_{\beta}(t)$ of base β.

In order to simulate data taken regularly, a random date t is chosen uniformly over an interval of ten years. After that, the value of $f(t)$ is calculated and an error (normal distributed) is added to it. The error is retrieved from a normal distribution $N(mean, sd)$ having $mean = 0$ and a standard deviation sd which varies during the experiment in order to see what happens when the values become sparse on the y-axis. To simulate data taken irregularly, a date t is randomly chosen from the exponential distribution with parameter λ. λ controls how concentrated, respectively sparse, the data are over the time. A small λ means that the data are more concentrated in one point (data are taken irregularly), a big λ instead distributes the data more sparse (the distribution is more uniform over time, data are taken more regularly), λ varies during the experiment in order to see what happens when data is collected more and more distributed.

For case 1, different datasets have been generated, choosing different lines (changing parameters a and b) or choosing different numbers of measurements n. Figure 7.2 gives two examples of such data. For case 2, the parameters a and b of the line are fixed as well as the standard deviation. The only parameter which changes is λ, Figure 7.3 gives an example of such data.

For case 3, different datasets have been generated as well, choosing different curves (changing the base β of the log function) or choosing different numbers of measurements n, in Figure 7.4 two examples are reported. In case 4, β is fixed as well as the standard deviation, the only parameter that changes is λ, Figure 7.5 reports an example.

Figure 7.6 evaluates case 1. Graphs (a) and (b) of Figure 7.6 present the results from data that have been simulated with a line having a small positive slope, while graphs (c) and (d) come from data that have been simulated with



(a) a=10, b=0.00000002, sd=0.6, n=100 (b) a=50, b=-0.0000001, sd=2, n=100

Figure 7.2: Synthetic data, distributed linearly and taken regularly

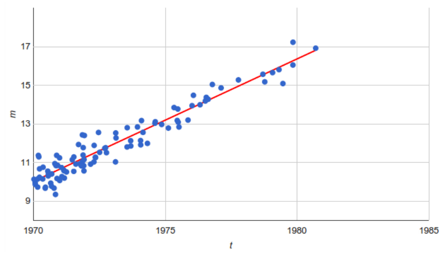
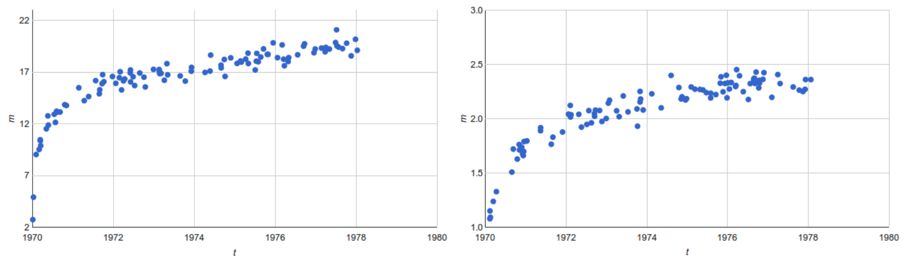


Figure 7.3: Synthetic data, distributed linearly and taken irregularly, a=10, b=0.00000002, sd=0.5, n=100, λ=2.5



(a) β=1.5, sd=0.6, n=100 (b) β=27, sd=0.08, n=100

Figure 7.4: Synthetic data, distributed logarithmically and taken regularly

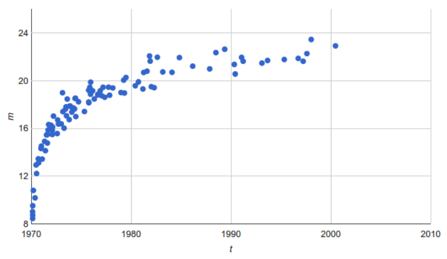
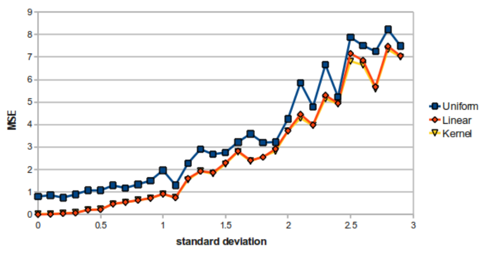
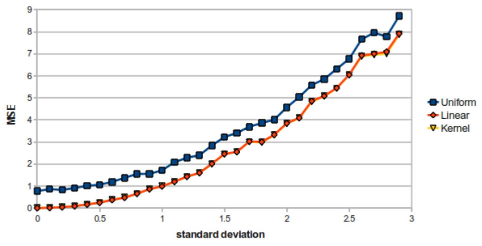


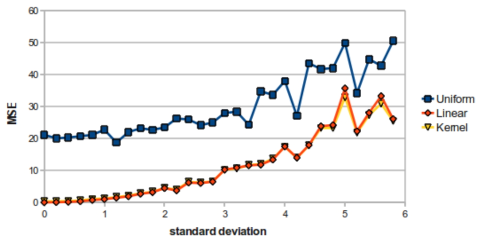
Figure 7.5: Synthetic data, distributed logarithmically and taken irregularly, β=1.5, sd=0.6, n=100



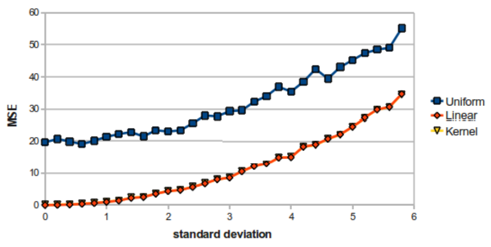
(a) n = 100, a = 10, b = 0.00000002



(b) n = 1000, a = 10, b = 0.00000002



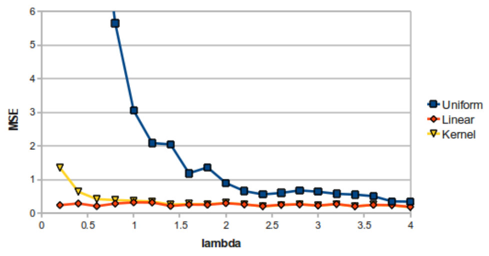
(c) n = 100, a = 50, b = -0.000001



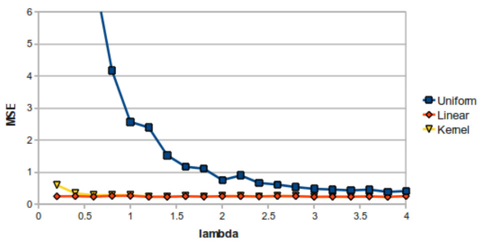
(d) n = 1000, a = 50, b = -0.000001

Figure 7.6: Case 1: MSE for synthetic data, distributed linearly and taken regularly

a line having a big negative slope. From these graphs we can clearly see how linear and kernel regressions always perform better than simply taking the mean value over the entire history. This observation works independently of the number of measurements, the slope of the line or the value for standard deviation. The difference of the performance between linear or kernel and the uniform functions also increases, when the slope of the line increases. As expected, the MSE grows for every fitting function, when the standard deviation grows. Since the error of fitting the data with a line is always the smallest one, the linear trend is always recognized and the summary is computed using linear regression.

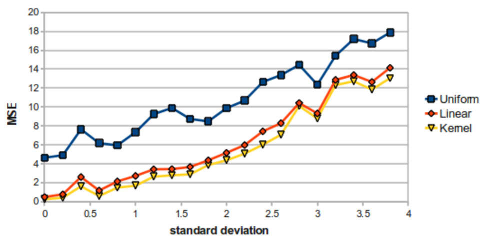


(a) n = 100, a = 10, b = 0.00000002, standard deviation = 0.5

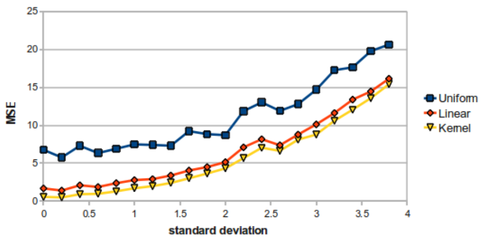


(b) n = 1000, a = 10, b = 0.00000002, standard deviation = 0.5

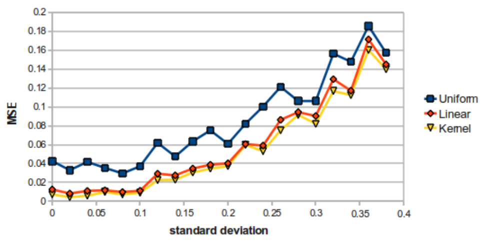
Figure 7.7: Case 2: MSE for synthetic data, distributed linearly and taken irregularly



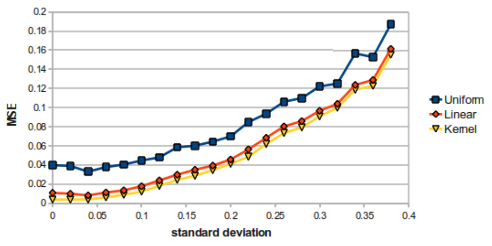
(a) n = 100, b = 1.5



(b) n = 1000, b = 1.5



(c) n = 100, b = 200



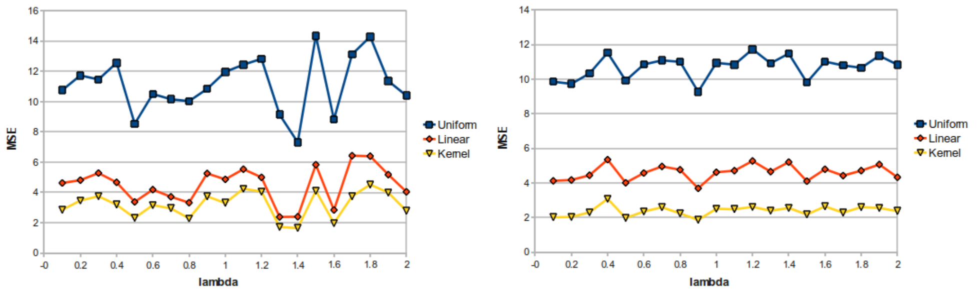
(d) n = 1000, b = 200

Figure 7.8: Case 3: MSE for synthetic data, distributed logarithmically and taken regularly

Figure 7.7 reports the MSEs for case 2. For very small λ (data taken more irregularly) uniform and kernel regressions perform much worse than the linear one. When λ becomes bigger, linear and kernel regressions produce almost the same MSE, while uniform function always remains slightly worse. It is very interesting to see how using linear regression makes the MSE remain almost constant for every value of λ. This means, that linear regression can deal very well with different data distributions over time (with data which follow a linear trend), while taking the mean value over the entire history can lead to big imprecisions, if data are taken irregularly. Similarly to case 1, the error of fitting the data with a line is always the smallest one, thus linear regression is always used to compute the up-to-date summary.

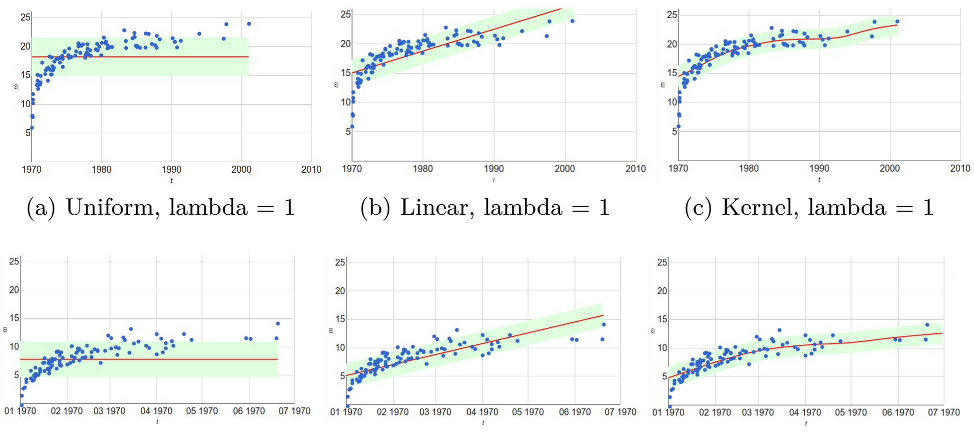
Figure 7.8 evaluates case 3. Like in case 1 of Figure 7.6 with linearly distributed data, the uniform function always performs considerably worse than the linear and the kernel ones, but this time the kernel regression, as expected, has always a smaller MSE than the linear regression for every tested standard deviation. These results show that kernel regression is better for fitting non-linear distributions.

Figure 7.9 presents the results for case 4. We can clearly see that kernel regression always performs much better than linear and uniform functions. In contrast to the results of the data that are taken irregularly and have a linear trend, the MSE remains constant for every distribution of the data over time (different λ parameters) and for every fitting function. This comes from the fact that the logarithmic curve becomes flat very fast, but still has a very steep shape at the beginning of the function generating the MSE difference between



(a) n = 100, b = 1.5, standard deviation = 0.8 (b) n = 1000, b = 1.5, standard deviation = 0.8

Figure 7.9: Case 4: MSE for synthetic data, distributed logarithmically and taken irregularly



(a) Uniform, lambda = 1 (b) Linear, lambda = 1 (c) Kernel, lambda = 1

(d) Uniform, lambda = 21 (e) Linear, lambda = 21 (f) Kernel, lambda = 21

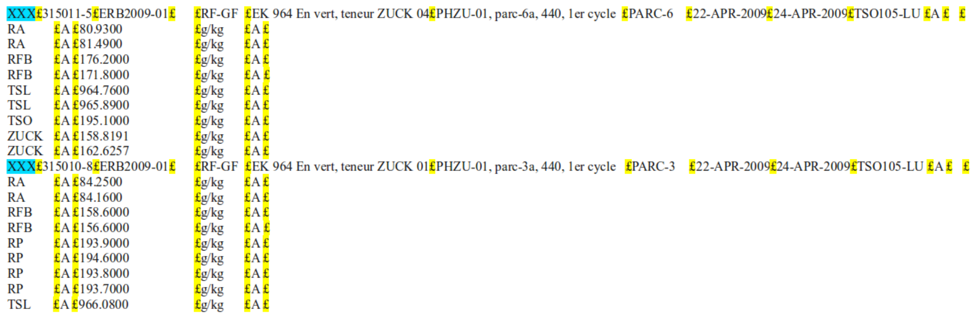
Figure 7.10: Synthetic data, distributed logarithmically, taken irregularly

the three methods. Figure 7.10 illustrates how the MSEs can remain constant also choosing very different λ (in this case 1 and 62). The y-axis has the same scale for every graph, so it is very easy to see how the RMSEs areas (light gray areas) just slide downward, practically without changing their width. At the same time the scale for the x-axis becomes smaller for $\lambda = 62$. This means that the data are more compacted, but essentially maintain the same distribution, managing thus to maintain the same MSE for every tested λ.

8 Data Import

This section describes the raw feed data and presents an application to import it into the database. Since no documentation exists about how the researchers store the raw data of the analyses, we went to Annelies Bracher, who is a researcher in Agroscope, and with her collaboration we recreated the whole process of generating a so-called LIMS file, where all the raw data are stored. A LIMS file is an export file from the LIMS system. The LIMS system is a software, which is used by the researchers of Agroscope to store the measurements derived from the chemical analyses of the feed samples.

8.1 Raw Data - LIMS File



```
XXX£315011-5£ERB2009-01£    £RF-GF  £EK 964 En vert, teneur ZUCK 04£PHZU-01, parc-6a, 440, 1er cycle  £PARC-6    £22-APR-2009£24-APR-2009£TSO105-LU £A £   £
RA      £A £80.9300         £g/kg   £A £
RA      £A £81.4900         £g/kg   £A £
RFB     £A £176.2000        £g/kg   £A £
RFB     £A £171.8000        £g/kg   £A £
TSL     £A £964.7600        £g/kg   £A £
TSL     £A £965.8900        £g/kg   £A £
TSO     £A £195.1000        £g/kg   £A £
ZUCK    £A £158.8191        £g/kg   £A £
ZUCK    £A £162.6257        £g/kg   £A £
XXX£315010-8£ERB2009-01£    £RF-GF  £EK 964 En vert, teneur ZUCK 01£PHZU-01, parc-3a, 440, 1er cycle   £PARC-3    £22-APR-2009£24-APR-2009£TSO105-LU £A £   £
RA      £A £84.2500         £g/kg   £A £
RA      £A £84.1600         £g/kg   £A £
RFB     £A £158.6000        £g/kg   £A £
RFB     £A £156.6000        £g/kg   £A £
RP      £A £193.9000        £g/kg   £A £
RP      £A £194.6000        £g/kg   £A £
RP      £A £193.8000        £g/kg   £A £
RP      £A £193.7000        £g/kg   £A £
TSL     £A £966.0800        £g/kg   £A £
```

Figure 8.1: LIMS file

A LIMS file is structured according to the samples: every row of a LIMS file which begins with "XXX" starts the description of a sample. While reading the description of a sample, it is necessary to distinguish the first row. The first row contains the meta-data, while the other rows describe measurements of the nutrients. Each row contains columns separated by the character "£". The descriptions of the columns of the first row are listed in Table 8.1, the descriptions of the columns of the rows containing a measurement are listed in Table 8.2.

Column number	**Description**	**Example**
1	Identification characters for the begin of a sample	“XXX”
2	LIMS number	“315011-5”
3	Project code	“ERB2009-01”
4	Always empty	“”
5	LIMS category of a feed	“RF-GF”
6	Free text to describe a sample (origin, feed type, etc.)	“EK 964 En vert, teneur ZUCK 04”
7	Free text to describe a sample (origin, feed type, etc.)	“PHZU-01, parc-6a, 440, 1er cycle”
8	Method of delivery (by mail, by hand, from the own farm etc.)	“PARC-6”
9	Sample date	“22-APR-2009”
10	Arrival date (at the laboratory)	“24-APR-2009”
11	LIMS abbreviation of the sample’s preparation method	“TSO105-LU”
12	LIMS status of the whole sample	“A”
13	Always empty	“”

Table 8.1: Columns of the first row of the sample description

Column number	**Description**	**Example**
1	Abbreviated nutrient name	“RA”
2	LIMS status of the measurement (completed, available, in preparation, etc.)	“A”
3	Measured value	“80.9300”
4	Unity of measure	“g/kg”
5	LIMS status of the measurement (approved, rejected, etc.)	“A”

Table 8.2: Columns of the other rows of the sample description

8.2 Data Import Application

This section describes a java application to import the raw data into the database. The application permits to read a LIMS file and directly import the measurements contained in it into the database. The code is about 250 lines long and is divided into three main methods.

1. As a first step the LIMS file is read row by row and for every row it is recognized whether the row is the description of a sample or of a measurement. After this the sample, respectively the measurement, is imported into the database using the appropriate method:

```
private void importData(File limsFile) {

    if (limsFile.hasNext()) {
        while (limsFile.hasNextLine()){

            String tempRow = limsFile.nextLine();
            tempRow.useDelimiter("£");
            String firstToken = tempRow.next();
            if (firstToken.startsWith("XXX")){
              importSample(tempRow);
            }
            else{

              importMeasure(firstToken, tempRow);
            }
        }
    }

}
```

2. With this method all the information about a sample is transformed into the SQL insert statement. If the sample is of a feed type that is not already present in the database, also the new feed type is inserted:

```
private void importSample(String line){

    //Feed category of the sample
    String feedName = line.next().replace(" ", "");
    int feedRef = -1;
    ResultSet results = executeQuery("SELECT feed_key FROM Feed" +
        "WHERE feed_name = \'" + feedName + "\'");
    while (results.next()) {
        feedRef = results.getInt("feed_key");
    }
    //If no such feed is found, then introduce it into the db
    if(feedRef == -1){
        st.execute("INSERT INTO Feed...);
    }
    //The line is divided into the remaining information fields
    //and a new sample is inserted into the database
}
```

3. With this method all the information about a measurement is transformed into the SLQ insert statement. If the measurement is done on a nutrient type that is not already present in the database, also the new nutrient type is inserted:

```
    private void importMeasure(String firstToken, String line){
        //Abbreviated nutrient name
        String nutrientAbbreviation = firstToken.replace(" ", "");
        int nutrientRef = -1;
        ResultSet results = executeQuery("SELECT nutrient_key FROM Nutrient + "

            "WHERE n_abbreviation = \'" + nutrientAbbreviation + "\'");
        while (results.next()) {
            nutrientRef = rs.getInt("nutrient_key");
        }
        //If no such nutrient is found, then introduce it into the db
        if(feedRef == -1){
            st.execute("INSERT INTO Nutrient...);
        }
        //The line is divided into the remaining information fields
        //and a new measurements is inserted into the database
    }
```

The main challenges importing the data from the LIMS files are:

- Since the format of the measured values is not strictly defined, everybody can write anything as a value for a measurement, inputs like "<0.001", ">0.0001", "~0.02" or "mehr als 5%" can be entered from the researchers as valid measurement values. For all these cases and many others we made the choice to eliminate all the invalid characters like "<" or "~" from the input, since having such characters would not give us the possibility to do numerical analyses on these values. Another possibility is to simply skip these measurements.
- In rare cases the sampling date for a measurement is missing, thus we had the choice to give a default date-value or to skip the entire measurement.
- Sometimes space characters are included in the "LIMS category of a feed" field. In this case all the space characters should be removed from this information field - if not, the names "GRASS" and " GRASS" cannot be recognized as a unique "LIMS category of a feed".

- In general, characters like “ or ’ should be removed or escaped in an appropriate way - if not, the SQL insert statement does not work correctly.
- The last challenge is inserting a new row into the database: since inserting the raw data row by row into the database is taking too much time, we decided to create a string, where the measurement values for approximately 200 samples could be stored and then to insert them in one go, decreasing the total insert time.

Processing the LIMS file provided by the Agronomic Institute and inserting every measurement into the database took about 25 minutes. The 334 Mb file contained data from approximately 110000 different samples on a total of about 1.2 million measurements. We do not import all the column present in the LIMS file, as column number 8 of the meta-data row, since is not used by the current application of the feed database.

9 Conclusions and Future Work

The poor quality of the available data makes the computation of up-to-date summaries a challenging task. However, despite this we managed to develop a method to compute up-to-date summaries, that is better than simply averaging over the entire history of nutrient measurements. With the help of linear and kernel regression, we are able to detect linear and non-linear trends and to compute values for the up-to-date summary, which are more close to the real world ones than aggregating over the entire history. The online computation of up-to-date summaries is, at the end of this work, a new functionality of the web interface of the temporal Feed Database, which can now be fed directly importing the raw data.

The experimental part of this thesis clearly evidences the better performances that linear and kernel regression always have in comparison to aggregating nutrient measurements over the entire history. Especially then, when the quality of the data decreases, the computation of up-to-date summaries using such an aggregation can lead to serious deviation from the real world state.

Although our approach achieved good results, more work remains to be done. First of all, intensive evaluations on real world data is needed to test the method used to compute up-to-date summaries. Second, in order to achieve even better results, data should also be fitted with other regression methods, for example using logarithmic and polynomial regression, cause we strongly believe that fitting data using regression functions is much more easy, time efficient and powerful than selecting a time interval to be aggregated.

Bibliography

[1] http://code.google.com/apis/chart/ (23.08.2011)

[2] http://maps.google.ch/maps/api/js?sensor=true (23.08.2011)

[3] http://www.agroscope.admin.ch (23.08.2011)

[4] http://www.google.com/jsapi (23.08.2011)

[5] S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology. ACM SIGMOD Record. (1997)

[6] H. Gregersen, C. S. Jensen. Conceptual Modeling of Time-Varying Information. (1998)

[7] A. Gupta, I. S. Mumick. Maintenance of materialized views: problems, techniques, and applications. MIT Press, Cambridge, MA. (1999)

[8] D. Moore, D. Neal, G. McCabe. The Practice of Business Statistics: Using Data for Decisions. W. H. Freeman, New York. (2003)

[9] P. Pin-Shan Chen. The entity-relationship model - toward a unified view of data. (1976)

[10] D. W. Scott. Multivariate Density Estimation. Wiley & Sons, New York. (1992)

[11] B. Silverman. Density Estimation for Statistics and Data Analysis. Chapman & Hall, London. (1986)

[12] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R. T. Snodgrass. Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings Publishing Company. (1993)