Facharbeit HS 2014
Institut für Informatik, Lehrstuhl Datenbankentechnologie
Universität Zürich

Visualization of nutrient/nutrient relationships

by
Benedikt Steger
10-735-520

Supervised by Francesco Cafagna
Research assistant and doctoral student

Table of contents

1. Overview

The Feed Base is a database of feed samples containing nutrients. One can filter various nutrients. The main result view consists of a table showing the amounts of selected nutrients for found feed samples. But an additional result view is needed in the Feed Base. The view should be able to show correlations between arbitrary chosen nutrients available in the database. Such an interface was built with the usage of a scatter chart of Google Developer Tools and normal HTML input elements. The calculation of linear regressions is now available. The user may define maximum data points to be shown by setting a limit in the advanced options. Because an attack on the server had succeeded, I have pen-tested the application.

2. The Setting

For the possibility to contribute to the code base of the Feed Base, I was provided with a copy of the files residing under the root of the webserver in production mode. Those files make following assumptions:

- The Feed Base is the only application on the webserver and therefore defines absolute links hard-coded derived from the root directory of the server.
- Apache and PHP are installed, and PHP's options define error displaying off.
- External resources used by the Feed Base, mainly Google's API for Google Maps and Google's visualization libraries, are only referenced online and are not included in the files delivered, but are accessible for everyone with an internet connection.
- The database is on an accessible server in the internet. Additional backups are in the /tfdb_backup folder.

As I own a computer for strict offline use as personal protection against infection of malicious software, of malware, of viruses and of other yet unknown possible attack vectors, I have a separate computer whose designated use is the access to the internet. The latter computer, however, has only an Intel Atom processor and lacks resources in terms of computational speed and memory, such that developing became extremely uncomfortable and really nearly impossible because of the very long waiting times.

As a consequence, I tried to build a cache of the requested external resources such as the Javascript files, the stylesheets and the graphics by using the World Wide Web Offline Explorer (WWWOFFLE). The cache should have been transferred to the computer without internet allowing an access to the previously mentioned resources that had been available only on internet before. The database would have been imported into a local instance of the Postgres Server from a provided backup. Early tests have shown, as a matter of fact, that resources were requested from a secure channel, disallowing the use of a proxy without much effort in configuring a decryption-encryption variant with signed certificates for the proxy.

Therefore my fast computer for offline use was made ready for internet. This meant moving out all files stored on disk, disconnecting from the LAN and installing a fresh installation of OpenSuse 13.1. The needed programs (Feed Base, Apache 2, PHP 5, kate, PhpPgAdmin) were installed. I had to work around the implicit assumptions made by Feed Base, including:

- Granting the root path for Feed Base alone. This issue was resolved by setting up a computer just for this job.
- Identifying non-responsive behaviour (blank screen) as a result of error reporting on the level notice and of actual errors (uninitialized variable not tested with isset()). As Feed Base

is chiefly Ajax-driven the error messages have been only visible in now broken Ajax response bodies. Those have only been detected by network-traffic analysis tools as provided by the developer tools of Firefox. I turned PHP's error reporting off, resulting in a more complicated development of relevant code for Ad-hoc testing.

- Establishing an internet connection solved multiple problems. On the other hand the requiring of an internet connection resulted in shorter development times, which means no mobility, and less setup possibilities.

Additionaly Firefox didn't trust Google Internet Authority G2 - an authority signing certificates for secure connections. I have trusted that authority in order to use the resource located on https://www.google.com/jsapi and in order to get around a blank screen. Therefore the Feed Base requests browser versions later developed than December 2013.

3. Implementation details

In order to implement the new functionality, I added a new panel labeled “Correlated nutrients” which contains four elements:

1. The selection box for the independent variable
2. The selection box for dependent variables allowing a selection for multiple nutrients.
3. The scatter plot itself
4. The warning shown whenever the result view is restricted. The warning includes the limit and total number of feed samples.

All line numbers in the code are defined in respect to the files I changed and submitted.

The content of the **selection box for the independent variable** is a copy of the HTML-element “select” with the ID “dd_charts_nutrient_a”, which is defined in /index.html:322 and filled by default because of executing the line in /layout_resources/jscripts/output_results-12.js:882:
`selected_nutrients.push(push_object);`
Whenever “dd_charts_nutrient_a” gets updated, because the user changes the query by defining other nutrients or feeds, or anything else, the contents of the selection box change, as a result, too. This happens because redrawCharts() is called by load_samples() in output_result-12.js:612. The selection box “dd_charts_nutrient_a” is the first box displayed on the panel “Scatter chart”. My selection box has the ID “dd_correlation_indep”. A change of the selection, including defining a selection at all, executes redrawCharts(), as I have implemented with a Javascript handler in index.php:328:
```
<select id="dd_correlation_indep" class="dd_select_n"
onchange="redrawCharts();">
```

The **dependent variables** are defined as a selection box for multiple selections, as defined with `multiple=”multiple”` in index.php:329 and has the ID “dd_select_dep”. The values are also copied from “dd_charts_nutrient_a”, as it is written in output_result-12.js, function load_samples(), line 443:
```
$("#dd_correlation_dep").html($
("#dd_charts_nutrient_a").html().substring());
```
Because of copying the very same list, the independent variable representing the nutrient on the x-axis, gets copied, too. As no scatter chart with only one dimension could be drawn or as a comparison of the correlation of a variable with itself is senseless, the status of selection of the independent variable is ignored - a process named “degrading gracefully” resulting in silently

ignoring an error for a better user experience. This behavior is visible in output_results-12.js:1004 while building the list of selections in the selection box with the dependend variables:
`if(parseInt($(this).attr("value"))!=independent_nutrient) { //...`

As soon as the selection of nutrients changes, the function **redrawCharts()** is called by the handler who is listening on “dd_select_dep” all the time. In the end, the scatter chart named “dd_nutrient_correlation”, which is introduced in index.php:342 and is fed with default values in output_results-12.js:473, the scatter chart gets updated. redrawCharts() must first fetch the selection of the panel called “dd_stat_type_select”. If this value is “6”, which is defined in index.php:319 and caught in output_results-12.js:966, first all other panels must be hidden. They are not found by any means, but rather have to be listed for each panel manually. Because hiding my panel is needed upon the selection of another one, I had to introduce following commands to hide my panel after each evaluation of expression, like shown in the following typical example (line 953ff):

```
}else if ($("#dd_stat_type_select").val() == "5"){
$("#dd_nutrient_correlation").css("display", "none");
$("#dd_correlation_select").css("display", "none"); // …
```

redrawCharts() is not only called by changing the selection, but also is used generally to update correctly the panels, whenever the state outside has changed. This includes an update of the selected nutrients, feeds, etc. Therefore, I have to test the number of rows returned by a probably newly executed query in order to show a warning when the result view is restricted, which is a requirement in the written task. This gets checked with the expression `query_result.getNumberOfRows()<number_of_feed_samples` on line 976.

I have introduced the function getSelectedIndepNutrientId() to fetch the ID of the selection choice made by the user for the independent value. This ID is needed for two additions to the chart:

1. To feed the push_object variable with the ID and the complete label, as defined on line 995.
2. To add the complete label as a title to the x-axis, as defined on line 996.

Google's visualization library reported that the first column is the one which defines the x-axis. Therefore, all further columns defined will represent the dependent variables. Since the selection is defined as “multiple”, I have to iterate through each list element, check whether it is selected, check whether it is not the independent variable and check that the ID is not a dummy ID like performed on lines 1001, 1002, 1004 and 1006.

The dependent data sets get attached to the selected_nutrients in the same manner as the independent variable was added, except that the label is registered in the legend (line 448, 1008). The dependent data sets additionally define a series-specific configuration in order to introduce the **trend line**, as specified in a requirement in the written task (line 1010):
`… .push({ type: 'linear', showR2: true, visibleInLegend: true });`
This line finally makes three very important changes:

1. Introduce the linear, derived regression trend line. There are other regressions possible, but a quadratic regression or a regression with another exponent was not requested. The color is changed automatically for each new series added.
2. Calculate and display the statistical value R^2, a measure defining, how well the data points correlate to the trend line. “0” means no correlation, resulting in an arbitrary line, whereas “1” means a perfect correlation: all points are on one line and define the line completely. As the requirements aren't explicit in this point, I justified the use of this value because the display of this value in a screenshot is explicitly shown in the written task. By the way, I am, however, not really an expert in statistical analysis of agricultural data.

3. visibleInLegend makes the derived regression equations and the R^2-value visible in the legend allowing further analysis.

Whenever an element was selected and wasn't the independent variable, a counter was increased (line 1011) in order to decide whether to draw a chart at all or not (line 1017). This leaves the area reserved for the chart white whenever the selection is illogical, resulting in a better user experience by hiding an ugly red error message otherwise shown.

The legend for the x-axis competes for space with the ticks and numbers explaining the x-values. The ticks and numbers are generated as part of the default settings defined by the maximum and minimum values along the x-axis. Introducing a title for the x-axis hid the numbers, which resided under the x-axis. I found no other solution for displaying both than by moving the numbers above the x-axis: `textPosition: "in"` (line 473). The numbers have a halo, making them readable regardless of the background painted by data points.

The list of dependent nutrients can get very long. Therefore, I wanted to put it on the right side of the panel, using the hole height of the panel. But I reasoned that a large chart is far more important than a large selection box, especially when the size of the panel is not a percentage of the hole browser window size, but a fixed size in pixels, so that the chart won't get bigger on a display with a higher resolution. Therefore, I have introduced the CSS rule `select.dd_select_dep` (main-12.css:547, used by index.php:329) defining a maximum height of 5 font sizes.

I have introduced and translated the strings “Warning: only” for introducing the limits and “Nutrient correlation” for the panel selection box. I reused the existing `$_terms["result_size_mes"]` (terms.de.php:107), allowing me convenient access to the limits (index.php:331).

output_result-12.js is built in such a way that thorough the file references to my panel are needed:

- Like discussed before, my panel needs an update to the display status whenever another panel changes.
- I had to instantiate my correlated_scatter_chart (line 27)
- I had to release my chart when requested (line 118)
- I had to introduce the variable correlated_chart_was_displayed (line 162)
- I copied the values from “dd_charts_nutrient_a”, as discussed before (lines 436-444)
- I had to create the scatter chart (lines 470-494)
- I had to hook into clearSelection() (line 800)
- A selection of a data point is handled in selectSampleOnChart(sample_id), too (lines 840, 841)
- I introduced getSelectedIndepNutrientId() resulting for a better readability of the code.

Every change can be found in the code thanks to the following identifier for changes before each instruction or block of instructions, which is inserted by me:
`//TODO: Beni`

Overall, no query was required to build as I have only added an additional result view fed with the data of the user query. As a result, all my changes are on the client side.

The edited files are /layout_resources/jscripts/**output_results-12.js**, /**index.php**, /layout_resources/lng/**terms_*.php**, and /layout_resources/css/**main-12.css**.

4. Derived regression equations' impact on performance

The scatter chart allows the definition of regression lines in the options of the DataView, which is

used to define the scatter chart. The additional computing time for those lines is dependent on four variables:

1. The number of series
2. The running time of the algorithm to calculate R^2, the slope and vertical position of the regression line.
3. The time needed to draw the regression line with the calculated parameters
4. The resources available on the computer running Javascript.

A line is composed of calculated points depending on the analyzed regression function parameters. Each point on the line is positioned in equal distance to its predecessor, forming a line of points with exact the same distance to each other.

First of all, all values must be read and taken into an equation in order to find out R^2. This includes summing up the values and taking the root of a sum twice. The time needed to find R^2 grows linearly with the amount of data points.

The running time for display:
Let **d** be the size/diameter of a dot in pixels. **w** is defined as the maximum possible width of the scatter chart, completely used in the worst case. w is defined in /index.php as 500px. Empirical analysis showed that the centers of the dots have a distance of d/2, resulting in overlapping dots by a half on both sides. Let **n** be the number of dots. This value can be derived from the following formula: **n**=w*2/d. Each dot occupies a short amount of the whole width, namely the length of its radius. The width defined isn't the width the scatter plot uses, because the function has to fit legends into the provided space, too. But for the calculation of the running time this is a constant to be ignored if the number of data points increases. **t**=2ms is the measured time needed by the computer to effectively draw a point onto the scatter chart. The running time grows linearly with the number of series **s** the user wants to compare (Example: 3). For each series, each dot needs some time to be drawn, therefore the resulting time equals to s*t*n. The time needed for display is 3*2*(500*2/4)=1500ms. As no exponentiation is performed, the running time for display is linear, too.

The overall performance is therefore proven to be linear in its nature. Despite the linear regression being a simple line, a lot of dots are drawn. This doesn't increase the complexity of the running time, but makes the linear function steeper. This means that the computer gets slower very quickly.

I tried to measure it empirically and “felt” an increased “sluggishness” as I added more and more series to the scatter chart. I measured an increase of ~500ms for each added series. The calculation of the regression and the time needed to draw the dots aren't the only things taking up time. The normal data dots need time, too. Overall, more than four series resulted in uncomfortable long loading times.

I postulate that the overall performance depends on three main factors:

1. The number of series wished to be displayed. Each series needs some time for the data points, for the calculation of the regression and for the points of the regression line
2. The number of results displayed for each series.
3. The performance of the computer of the client, as all is performed on client side.

As the main factor contributing to the total running time is the time needed to draw a dot, and since this process is a slow one, introducing a regression line was a significant slow-down of felt responsiveness. But an added regression line took the same amount of time as drawing a new series

without regression line. Therefore, one can say, that the impact on performance isn't significantly derived from calculated correlations and regressions, which is quite fast, but chiefly from the time uncompiled and slow Javascript needs for drawing a dot.

If the user updates the chart seldom and wants to perform extensive analysis, he will readily take on some time. A user without patience, however, will have to decrease the number of data sets returned by the server in the advanced options, minimizing the number of points displayed in the scatter chart. Alternatively, he will have to decrease the number of series to be shown in order to increase responsiveness.

5. Security

5.1. Easy measures taken as a good security practice

Although the Feed Base hasn't much responsibilities towards its paying users, as written in the „Nutzungsbestimmungen“ under §6, a loss of the service due to a successful destroying attack or due to disallowed copying of valuable information is unacceptable. Some measures are easy to take and as a result provide a great increase in security. It is clearly impossible to defend against a targeted attack, explained by the law of diminishing returns, but at least some protection against automated scanners is a good way to protect oneself against a large percentage of possible attacks. A little protection gives good returns because there is profit included in the project and because modern malware like CryptoLocker has a strong destroying nature. As I have worked with the code and as an attack succeeded indeed, I have identified several attack vectors and I have proposed the counter-measures needed to take, as far as I have judged. One should not forget that other attacks may come up with other yet unknown ways to attack a system, even though one can't think of a new way anymore. Therefore, critical analysis of the code of as many people as possible or of an external certification would result in an effective increase in security. Unfortunately, public facing files are stored on the server, but shouldn't be. This counts especially for the backup under /tfdb_backup/latest_new.sql or for old files still residing in the public-accessible area.

5.2. Pen-testing: Identified vulnerabilities in The Feed Base

a) Rainbow-Table password attack

Affected files: /admin/user_management/user_management.php (line 31)
Severity: Middle
Prerequisites: A copy of the table „users“ of the database, for example, from a stolen backup. Or a hash gathered from a login.
Attack: Build an index of input values and their corresponding hashes calculated with the MD5-algorithm. Reverse look up each hash in the stolen list or the stolen hash from the login against hashed input strings, resulting in the password in clear text if a match is made.
Counter-measures: Salting, seldom used hash functions and iterations. Introduce a new field “salt” with a long random string, which gets appended to the password before hashing. This requires the attacker to build large databases producing prohibitive costs. Also, iterate the hashing process many times: 4069 is a good trade-off as in year 2015. Don't use MD5, use a longer hash like SHA512 or a seldom used one like Whirlpool in the iterations. Don't make backups publicly accessible.

b) SQL-Injection

This attack vector consists of a whole range of attacks. The simplest one is unfortunately the one succeeding in this case. The security is greatly improved, however, if the input values and variables used in the query definition are **SANATIZED**. The main problem consists in the misplaced trust in

requests from the legitimate server user requesting data from the database. One should not assume that unpublished source code hinders building up knowledge on the attacker side as the interface consisting of URLs in Ajax calls can be revealed by either analysis of the sent Javascript or normal traffic analysis of interaction performed by a legitimate user. See /layout_resources/jscripts/output_results-12.js:284 for an example. Security by obscurity actually lowers the level of security because less people will go through the code and because the programmer making wrong assumptions takes less steps in increasing protection, which is dangerous because reverse engineering is performed at a frightening efficiency. Unfortunately, the POST-parameters in most cases correspond to the database column names, revealing the database structure. The normal string ";--" at the end of the query is recommended, because pg_query() allows several commands per function call. Side-channel attacks as timing the answer times are nothing in relation to the plain SQL-injection possible. $_POST isn't safer than $_GET, even if it's harder to introduce an attack value than for simpler $_GET.

Affected files: At least /db_scripts/*, for example ajax-pg-query-where.php:48 (because of line 43)

Severity: Critical - automated scanners, programs for script-kiddies exist!

Prerequisites: Notion of the database structure
Attack: Complete or alter a normally legitimate SQL query during query buildup phase using altered values of used variables. The variables $_GET, $_POST, $_COOKIE and $_REQUEST are used and alterable. As an example, one could login successfully with foreign variables $_POST['un']="admin" and $_POST['ps']="' OR username='admin';--" in user_login.php without knowing the password! The injection can become as powerful as SQL, especially when arbitrary queries are allowed as in /data/ajax-pg-query.php and working with current configuration.
Counter-measures: Sanatize: `intval($numeric_variable)`, `preg_replace("/[^a-z, ,A-Z]/","",$string)`, etc. Better sanatize a variable multiple times, for example during initialization (`$numeric_variable=intval($_GET['qid'])`) and in the query (`...where id=".intval($numeric_variable)`). As for /data/ajax-pg-query.php allowing execution of arbitrary SQL queries: Separate between production environment and development environment, or at least enforce authentication.

c) Too much provided information

Affected files: /info.php (line 2)
Severity: Middle
Prerequisites: None
Attack: Assess the version and variables of PHP meant for development in order to identify attack vectors (version, register_globals==?). Known former attacks for installed version of PHP can be used for preparation. Although the real production server was last patched the 18th of December 2014, such information is easy to protect and harms, too.
Counter-measures: At least protect the access with a password: `if(isset($_GET['pw']) && $_GET['pw']=="vai3Fah1ew1waegh+iev"){phpinfo();}`

d) Weak passwords, missing encryption and hashes usable for logins

Affected systems: Webserver, user management of Feed Base, /db_scripts/user_login.php: 7, 10, 13, 23, /index.php: line with `Crypto.MD5(...`
Severity: Critical
Prerequisites: None
Attack: With a list of most commonly used passwords (adm[...]23), try each for the root/admin. Try the user name as password itself. Or just listen to the traffic and record the hashes to perform a successful replay attack, because only the hash of the password is needed for authorization.
Counter-measures: Use long passwords and strong encryption with HTTPS. I propose at least 20 characters and 8192 bit encryption. Additionally use a two-factor authentication. Users should use a

password manager in order to keep the password policy acceptance high. Set a default long password upon registration and don't send the password via E-Mail. Unset global Javascript variable `user_pass`. Because protection against a MITM attack is not an easy counter-measure, only calculate an iterated hash with the communicated salt on client side (see **a))** and encrypt with a secret agreed upon with the Diffie-Hellmann protocol before sending over the network. Then the server can decrypt the session payload and compare the hashes calculated (see **a)**) in order to grant authorization.

e) XSS (Cross-site scripting)

Affected files: /db_scripts/display_queries.php:62 because of /db_scripts/save_query.php:9

Severity: Middle

Prerequisites: An attacker knows one user's credential or gets a signed-in-user to perform a POST-request to save_query.php per Javascript. This can happen on a site with the same domain name, except the FQDN, with an identical layout or after the attacker added malicious Javascript code for each user in the stored query names. Or the attacker even logins a web-user due to the injection described under **b)** first, relieving the need of a signed-in-user.

Attack: Instead of writing the wished value of a variable, due to assuming another context, a wrong content is written finally. In the example, a foreign $_POST['den'] gets into the database, which is harmless for database unless part of a SQL-injection, but this now trusted source gets displayed 1:1 in the browser upon listing in display_queries.php. As attacker fetch the global Javascript variable user_pass still available from the login and place it as a parameter in a web bug, a little image residing on a C&C server, to collect hashes for an attack in the style of **a)**. Arbitrary HTML is useful for phishing, since the URL is correct.

Counter-measures: Sanatize your input. Escape <, >, & and “ with < > & and &quot.

f) Globally used variables

Affected files: /db_scripts/db_info.php

Severity: Middle

Prerequisites: Attacker can run PHP scripts on the server, or manages to get eval(ed).

Attack: The attacker returns values of variables never meant to be sent, for example $username, $password, $database, $host, and $port.

Counter-measures: Use previously set variables instantly and unset() those. Use db_info.php to produce a single $conn. Fortunately I haven't found an eval(), but I haven't checked for writable cache directories (directory with Excel files?), where evil files could get uploaded to, because my copy has secure directory permissions per default.

g) Administration area left open

Affected files: /admin/*

Severity: Critical

Prerequisites: Guess the path /admin correctly

Attack: Visit /admin and insert new users, new nutrients, get a list of backups or learn a lot of the database structure.

Counter-measures: Enforce authentication, e.g. if($_SESSION['user']=='admin') {...}.

h) Backups are not backups

Affected files: /tfdb_backup/*

Severity: Middle

Prerequisites: Attain user privileges on the web server.

Attack: Encrypt or delete all possible files on the webserver.

Counter-measures: A backup is only a backup, if it is not mounted on the system which contains the backed up data. Disconnect the backups physically.

6. Summary

The Feed Base has - thanks to my new implemented functionality - a way of displaying the results in a quickly apprehensive form of data visualization: the scatter plot of freely definable correlated nutrients. Additionally, regression lines are displayed for each dependent variable. The additional time needed to perform a correlation and a regression wasn't found to have significant impact, but there is still a loss in performance because of Javascript's slow drawing capabilities. Three critical security flaws are identified and solutions are proposed.

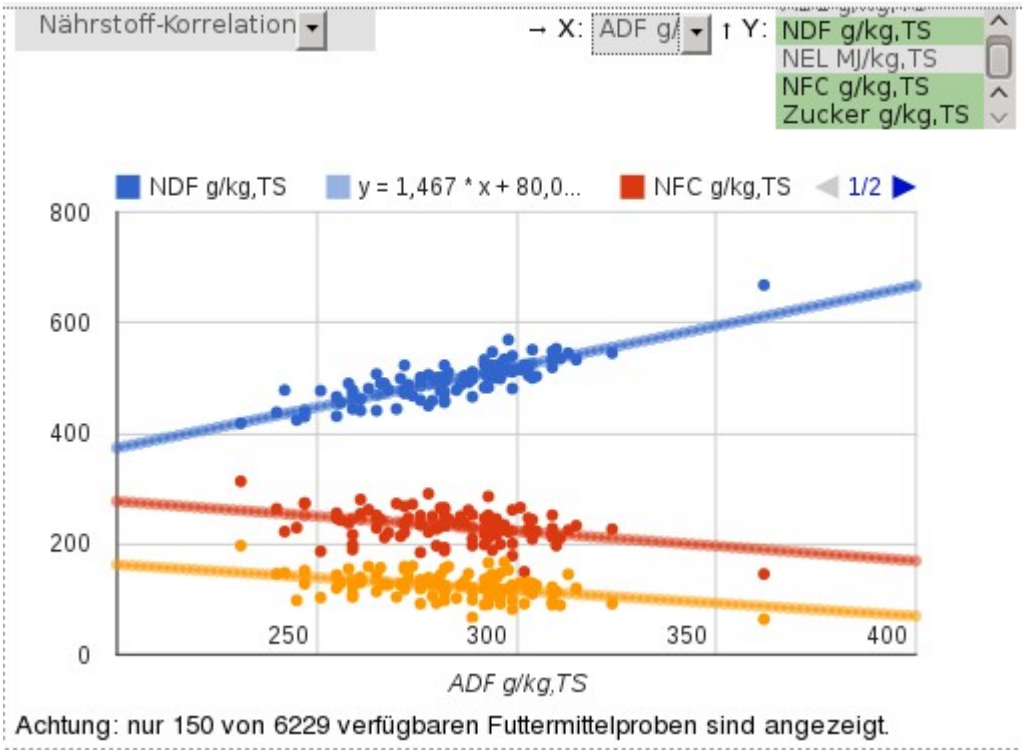

Figure 1: The final result of my new implemented functionality: the correlated nutrients

7. Expression of thanks

I really do thank Francesco Cafagna who helped whenever requested by answering questions and by providing help. This counts especially for the help provided during the adventurous installation of a copy of the complex product Feed Base.

8. Personal declaration

I hereby declare that the material contained in this Facharbeit is my own original work. Any quotation or paraphrase in this Facharbeit from the published or unpublished work of another individual or institution has been duly acknowledged. I have not submitted this Facharbeit, or any part of it, previously to any institution for assessment purposes.