Bachelor's thesis

Degree Programme in Engineering

2018

Timo Jokela

DETECTING EXCESSIVE TONER USAGE IN A PRINTER FLEET

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES
Degree Programme in Engineering
2018 | 45 pages

Timo Jokela

DETECTING EXCESSIVE TONER USAGE IN A PRINTER FLEET

The objective of this thesis was to provide both a theoretical and software solution for monitoring toner usage in a printer fleet. In particular it strives to offer the means to automatically detect and identify the devices with excessive toner consumption in relation to the bulk of the fleet. The sample device set with which the results are evaluated consists of color based printers under the brand name Sharp but the principles are readily applicable to all devices providing adequately detailed usage reports.

The methods devised rely on elementary machine learning strategies and simple bivariate statistical analysis and are implemented using the programming languages Python and Octave. Python is used for data extraction and presenting the results to the user, Octave for numerical analysis and algorithm development.

The results consist of visual analysis of the fleet and a multiparameter prediction algorithm known as the hypothesis. The parameters of the hypothesis are learned during the process. The solution as such is the most usable for fleets which undergo no sudden major changes in the fleet, but can be extended to other types of situations also.

KEYWORDS:
Data analysis, machine learning, MFP, multi-functional printer, toner usage

OPINNÄYTETYÖ | TIIVISTELMÄ
TURUN AMMATTIKORKEAKOULU
Insinööri (AMK)
2018 | 45 sivua

Timo Jokela

DETECTING EXCESSIVE TONER USAGE IN A PRINTER FLEET

Tämän opinnäytetyön tavoitteena on tarjota sekä teoreettinen että ohjelmistollinen ratkaisu tulostuskaluston värinkulutuksen seurantaan. Erityisesti se pyrkii löytämään keinot tunnistaa ja yksilöidä laitteet, jotka kuluttavat liiallisesti väriä verrattuna kaluston valtaosaan. Esimerkkijoukko, jonka pohjalta tuloksia arvioidaan, koostuu Sharp-tuotemerkillä valmistetuista värimonitoimilaitteista, mutta samat periaatteet ovat helposti sovellettavissa kaikkiin riittävän yksityiskohtaista käyttöraportointia tarjoaviin laitteisiin.

Kehitetyt menetelmät pohjautuvat alkeellisiin koneoppimisen strategioihin sekä yksinkertaiseen kaksimuuttujaiseen tilastolliseen analyysiin, ja ne on toteutettu Python- ja Octave-ohjelmointikieliä käyttäen. Pythonia käytetään datan eristämiseen sekä käyttäjälle esittämiseen, Octavea numeeriseen analyysiin sekä algoritmin kehittämiseen.

Tulokset koostuvat kaluston visuaalisesta analyysista sekä moniparametrisesta ennustealgoritmista, niin sanotusta hypoteesista. Kyseiset parametrit opitaan koneellisesti prosessin aikana. Ratkaisu on parhaiten sovellettavissa kalustoon, johon ei kohdistu yhtäkkisiä suuria muutoksia, mutta se voidaan laajentaa koskemaan myös muunlaisia tilanteita.

ASIASANAT:
Data-analyysi, koneoppiminen, monitoimilaite, tulostin, värinkulutus

CONTENT

APPENDICES

EQUATIONS

FIGURES

LISTINGS

PICTURES

ABBREVIATIONS AND GLOSSARY

Anomalous sample	An MFP with comparatively excessive toner usage
Bias term	A zero-condition term of the hypothesis
BW	Black and white
CDF	Cumulative distribution function
CMYK	Cyan, Magenta, Yellow and Key (black)
Cost function	A function for evaluating prediction errors
Decision boundary	The boundary separating samples with different classes
Feature	A statistical characteristic present in all of the samples
Gradient descent	An optimisation algorithm for finding the minimums
Greyscale	Halftoned BW to produce shades of grey
Label	A label assigned to a sample after classification
Hypothesis	The algorithm predicting the anomality of the sample
Learning rate	The rate of change in gradient descent
Logistic regression	A regression model with a binary-valued output
MFP	Multi-functional printer
PCC	Python Copy Counter, an MFP monitoring program
PDF	Probability density function
Sample	A combined statistical reading of a single device
UI	User interface

1 INTRODUCTION

The business of providing copiers and MFP devices (incorporating scanner, printer and copier functionality) and related services is often referred to as *office technology*. The common operating model is for the customer to lease the device itself and then agree on a service contract with the dealer. Typically such a contract states that in exchange for printer toner and maintenance services the customer is charged a per-page cost for the number of printed pages during a billing cycle.

The cost of a single page (measured in the A4 size in Finland) is different for pages in color and in black and white (or greyscale). This is because BW printing employs fewer resources than printing in color. The colors are typically of the CMYK[1] variety, and whereas a BW page is printed with only one toner (K), a color page uses all of them (K for pure black, CMY for producing other colors). Other cost-differentiating factors also exist, but the details have no relevance for this study.

A standard usage of a single color is 5 % per page [4]. That is, if all of the specific toner on a standard page were to be printed on a uniform region, then 5 % of the page would be covered. Hence, the page-specific costs are usually defined using this standard coverage and a considerable deviance from this standard use results in the device operating on loss. It is therefore a common practice to include a clause in the contract which makes it possible to place additional charges for usage exceeding the standard, even though these clauses are rarely applied.

However, regardless of contractual minutiae the objective of this thesis was to develop a method for both detecting devices using excessive amounts of toner and analyzing how common this excessive usage actually is. The techniques to achieve this objective were basic mathematical analysis coupled with the use of mathematically suited programming languages. The work will proceed from describing the initial setting to prototyping and finally implementing the solution, and will involve using an existing database of devices and their statistics. The primary focus of this thesis is on the application of a regression model called **logistic regression**, which is a method for binary classification of samples; that is, given two sets, to which does the current sample belong.

[1] There are other varieties such as the CMYKW model for the additional white color, but this is rarely used in office printing

It should also be noted that in descriptions relating to the operation of the devices and the business external references are not included. Instead, these parts of this thesis rely on the author's personal expertise stemming from five years of working in the field, in several technical roles such as maintenance and repair, pre-sales and as a technical specialist. Additionally, only superficial information of printing technology is needed, which works more as a background than as an essential element of the actual engineering work in this thesis. The method devised here is applicable even though the technicalities or the cost formation differ from the stated.

Why the effort?

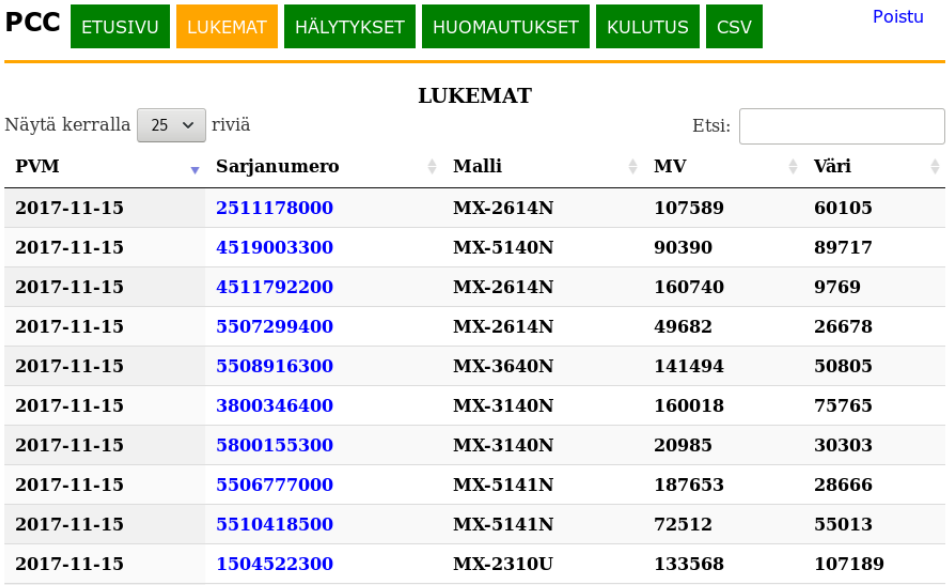
The reader might be inclined to wonder why make the effort to analyze the data in order to make a classification of excessive toner use. That is, why implement a mathematically encumbered model instead of just manually defining a limit of acceptable use. For this there is no clear-cut answer.

The essential thought behind this thesis, however, is that careful analysis provides the benefit of knowing just how the customer and device base is distributed in relation to toner usage, and helps avoid detecting the bulk of the customer base as excessive users. That is, even if a customer uses more toner than allowed as per the service contract, we do not want to consider this as excessive if it is within ”normal” limits. The handling of a customer relation is always a delicate matter, and an accusation of not keeping within the contractual limits is always bound to generate some bad will. We want to focus our attention to the group of devices inducing the largest losses.

2 BACKGROUND INFORMATION

An office technology dealer Oy Perkko operates mainly in the area of Southern Finland. For a large part of their history, the brand name Sharp has been the primary MFP product of the company. The dominant procedure in which Sharp devices report their statistics, such as the number of printed pages, is automatically sending timed status messages via email. Traditionally these messages

have been handled manually, with hired personnel entering relevant fields of information in the emails into the ERP system of the company. This is error-prone and tedious work as there are hundreds of devices reporting monthly. To solve this problem a system titled Python Copy Counter, or PCC, has been previously implemented by the author. This system uses the Python programming language and the Django framework to automatically extract data from arriving status messages, which is then inserted into an SQLite3 database. The gathered information is presented to the user in a dynamic web service exposed on the company intranet.



Picture 1: The PCC UI[2] in Finnish

In Picture 1 a fraction of the device fleet is visible. The information in the columns from left to right is the date of reporting (PVM), the serial number of the device (Sarjanumero), the model of the device (Malli), the total count of BW pages (MV) and the total count of color pages (Väri).

The statistics of a singular device can be more closely examined by clicking the serial number. This takes the user to a device-specific view in which the information from the monthly reports is presented in a tabular fashion. In addition to the previous, toner usage information is also presented whenever possible (if the device in question is of an adequately recent model).

Older Sharp devices report the toner usage in intervals (Picture 2), whereas the current generation of devices reports continuous change (Picture 3).

[2]Dynamic tables by Datatables, https://datatables.net/

K%	C%	M%	Y%	K#	C#	M#	Y#
0-25%	25-50%	50-75%	25-50%	2	2	2	1

Picture 2: Toner levels as intervals.

K%	C%	M%	Y%	K#	C#	M#	Y#
76%	45%	43%	40%	2	1	1	1

Picture 3: Continuous toner level changes.

The percentage signs in Pictures 2 and 3 display the levels of toner remaing for each color, and the hashtag (#) marked values display the total number of inserted cartridges. The situation is further complicated by the fact that different groups of devices use different types of cartridges and different types of cartridges have different kinds of yields. This has to be accounted for when analyzing the general situation.

3 DATA EXTRACTION

The PCC database is used as an example when retrieving and extracting data. However, this specific database is in no way a requirement in the process. Any form of storage allowing read access will do, be it an SQL database or a (humongous) CSV file. In addition, the algorithms can be applied to MFPs other than Sharp if the toner level and print count handling is adjusted accordingly.

The PCC database holds different tables for devices and status messages. Each status message is linked to a specific device with a foreign key named **device_id**. In addition to other information, the table for status messages (**pcc_total**) contains the following fields (all counted from the time of deployment):

- total_bw, total number of printed BW pages
- total_col, total number of printed color pages
- k_level, the approximate level of toner k remaining in the cartridge
- c_level, m_level, y_level
- k_installed, the number of installed K cartridges
- c_installed, m_installed, y_installed

We now proceed to extract relevant data from the database. The extraction program, fully listed in Appendix 1, is also written in Python but separated from other functionality of PCC. As this phase is required only when the final parameters are re-learned, it is both possible and preferable to execute it under supervised circumstances. Therefore, the Django convention for retrieving entries from the tables is omitted, and a standard SQL query is relied on instead.

```
SELECT serial_number, model, total_bw, total_col,
k_level, c_level, m_level, y_level,
k_installed, c_installed, m_installed, y_installed
FROM pcc_total INNER JOIN pcc_device
ON pcc_total.device_id = pcc_device.id WHERE c_installed > -1
ORDER BY pcc_device.serial_number, pcc_total.date_sent
```

Listing 1: SQL query for retrieving data from the PCC database

In Listing 1 we query the database for serial numbers and models and for all reported total counts, toner levels and installed cartridges. We impose one requirement in the **WHERE** clause, namely that of the number of installed cartridges actually included in the report. For both old devices and BW printers the number of installed CMY cartridges is represented as a value of -1, and we exclude the devices from the data to be analyzed.

For the yields of different cartridge types to be accurately accounted for, we define yield groups in the form of a Python dictionary. For example, the models **MX-2310U** and **MX-2314N** use the cartridge type **MX23**, which has different yields for K and CMY cartridges. Therefore, we first define how many standard 5 % pages can approximately be printed with a use of 1 % of total toner in the cartridge.

```
BK23 = 180      # one cartridge of MX23GTBA (black) gives approx. 18000 pages
COL23 = 100
```

Listing 2: Defining yields for 1 % of toner of the type MX23

This is shown in Listing 2. We then assign these as list values for the key that represents proper device models (Listing 3).

```
yields = {
    'MX-2310U': [BK23, COL23],
    'MX-2314N': [BK23, COL23],
```

Listing 3: A dictionary relating a model with a cartridge type

We also define a function in Listing 4 for parsing toner levels stored in the database fields.

```
def levels_to_ints(stat_line):
    stat_line = list(stat_line)

    # keep track that the initial levels were intervals or continuous
    stat_line.append(['interval' if '–' in stat_line[I_KI] else 'continuous'])

    for i in level_index:
        # older machines report toner levels in intervals 75–100%, 50–75% etc
        # we take the lower limit and add half (rounded down) to approximate
        if '–' in stat_line[i]:
            stat_line[i] = int(stat_line[i][0:stat_line[i].index('–')])+12
            stat_line.append('interval')
        else:
            stat_line[i] = int(stat_line[i].strip('%'))

    return stat_line
```

Listing 4: Function for parsing toner level strings

The actual work is completed next. We loop to obtain one record at a time. For each device (differentiated by the serial number), we are actually only interested in two records, the first and the last. In the interim we execute certain checks to see if the values show proper change. When we encounter the first record of the device, some values are initialized as shown in Listing 5 (shown for K, others omitted for brevity).

```
# the initial numbers of printed pages
bw_first = new_entry[I_BW]
col_first = new_entry[I_COL]

# initial % levels of toners
k_level_first = new_entry[I_KI]
...
# initial number of installed cartridges
k_ctrg_first = new_entry[I_Kc]
...
```

Listing 5: Setting the initial values of the device (truncated for CMY)

Values in the list **new_entry** have already been parsed with the function in Listing 4. After reaching the final record for the device currently inspected, the first and the last record (named **previous_entry**) are used to compute the differences in Listing 6.

```
# previous_entry is the last entry for the device now analyzed

# change in the number of printed pages
bw = previous_entry[I_BW] - bw_first
col = previous_entry[I_COL] - col_first

# change in the installed number of cartridges
k_crtg = previous_entry[I_Kc] - k_ctrg_first
...
# change in toner levels
k_level = k_level_first + 100 * k_crtg - previous_entry[I_Kl]
...
```

Listing 6: Computing the change for the device (truncated for CMY)

After this we are ready to set the final values for the device. In Listing 7 the attempt is made to determine the standard yield for each toner consumed during the lifetime of the device. In other words, the program computes how many standard pages should have been approximately printed with the total toner usage of that particular color. These values are inserted into a list as a dictionary, which is then exported as a csv file, which will be used in the following section to learn the parameters of the logistic regression model.

```
try:
    bk_yield = yields[previous_entry[I_MOD]][0]
    col_yield = yields[previous_entry[I_MOD]][1]

    # consumed toner in standard pages (5 % coverage)
    std_k = k_level * bk_yield
    std_c = c_level * col_yield
    std_m = m_level * col_yield
    std_y = y_level * col_yield

    # a dict later exported into a csv file
    stat_dict = {
            'bw': str(bw),
            'col': str(col),
            'std_k': str(std_k),
            'std_c': str(std_c),
            'std_m': str(std_m),
            'std_y': str(std_y),
            }

    device_stats.append(stat_dict)
except KeyError:
    # no yield defined for the (too old) model , drop it
    pass
```

Listing 7: Setting the final values of the device

4 INITIAL ANALYSIS OF THE DATA

For the logistic regression model to work, labeled data is needed. That is, we need to know which samples are considered anomalous (excessive in toner usage) and which ones are not. This is currently not the situation. Therefore, the individual samples must be labeled as normal or anomalous, preferably with an adequate level of cohesion, as random labeling is as good as no labeling at all. The analysis of the data is executed in the (largely Matlab compatible[2]) GNU Octave environment extended with the statistics package[7].

4.1 Visual analysis

First, the data gathered in the previous section is read from a csv file and for each sample an average use of C, M and Y toners is computed. By averaging CMY use we make the data easier to visualize. It is also more worthwile for the dealer to focus on the average use instead of searching for large uses of singular color.

```
1  % read data csv => matrix M
2  M = csvread('stats.csv');
3
4  % matrix A for the initial inspection of the data
5  A = zeros(size(M,1), 2);
6  for i = 1:size(M,1),
7    % average use of CMY toners
8    cmy_std_avg = mean(M(i,4:6),2);
9
10   % ratio of standard pages (5% coverage) to actually printed pages
11   k_std_to_real = M(i,3) / (M(i,1) + M(i,2));
12   cmy_std_to_real = cmy_std_avg / M(i,2);
13
14   % ratio of actually printed pages to standard pages
15   k_real_to_std = (M(i,1) + M(i,2)) / M(i,3);
16   cmy_real_to_std = M(i,2) / cmy_std_avg;
```

Listing 8: Insterting the data into a matrix

As shown in Listing 8, line 11, the ratio of standard pages to real pages is then calculated for color K. We take the estimated number of standard greyscale pages and divide it by the sum of actually printed greyscale and color pages (as color pages, too, can use toner K). The end result is essentially how many pages should have been possible to print in relation to how many pages were actually printed.

We conduct the same for the average color use (for color pages only, as CMY is not used in greyscale). On lines 15-16 we compute the inverses.

In order to visually analyze the data on a 2D graph, the following decisions are made:

- the origin of the graph represents the ratio 1:1, i.e. the point at which actual toner usage meets the standard
- an actual usage less than the standard has a negative signum

With these guidelines in mind, the larger of both ratios is chosen and adjusted by ± 1. The result is that the ”ideal” (1:1) usage is shifted to the origin and multiplied by 100 to obtain the percentages.

```
% pick wichever is larger, and adjust so that std>real equals moderate toner use
% (mark negative) and real>std equals excessive toner use (mark positive), also the
% ratio 1 equals to standard so shift to zero (+/- 1)
 if k_std_to_real > k_real_to_std,
   k = k_std_to_real - 1;
 else,
   k = -k_real_to_std + 1;
 end;

 if cmy_std_to_real > cmy_real_to_std,
   cmy = cmy_std_to_real - 1;
 else,
   cmy = -cmy_real_to_std + 1;
 end;

 % finally convert to percentages
 A(i,1) = round(k*100);
 A(i,2) = round(cmy*100);
end;
```

Listing 9: Organizing the data around the origin

The end result is an m-by-2 matrix A, m representing the sample set size (the number of devices in the set). The first column of A represents percentages for color K, the second for the averaged colors CMY. A, when plotted, visualizes the toner use in relation to the standard, as shown in Figure 1. The columns of A are named as the **features** of the data set.

In order to better understand the distributions the histograms are also plotted. In Figure 2 it can be seen that for both colors K and CMY the distribution is

roughly Gaussian, albeit slightly skewed to the left on the CMY side. In the current case this rough resemblance is however enough, as admittedly labeling data this way will always be somewhat arbitrary. In case either of the histograms would demonstrate considerably non-Gaussian characteristics, we would have to try additional manipulation, such as squaring of the samples [6].

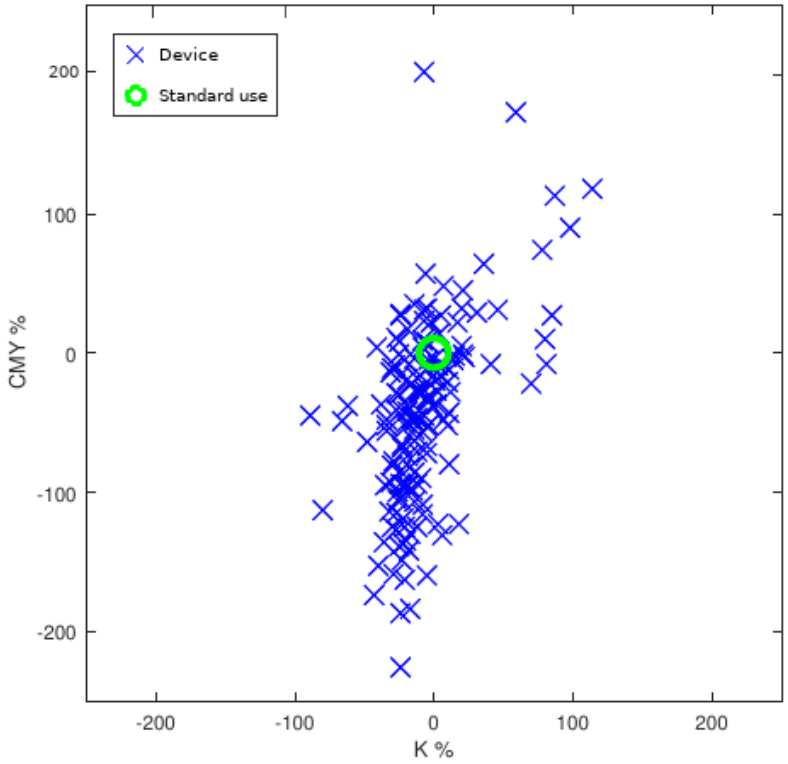


Figure 1: Toner usage of each device on a K/CMY chart

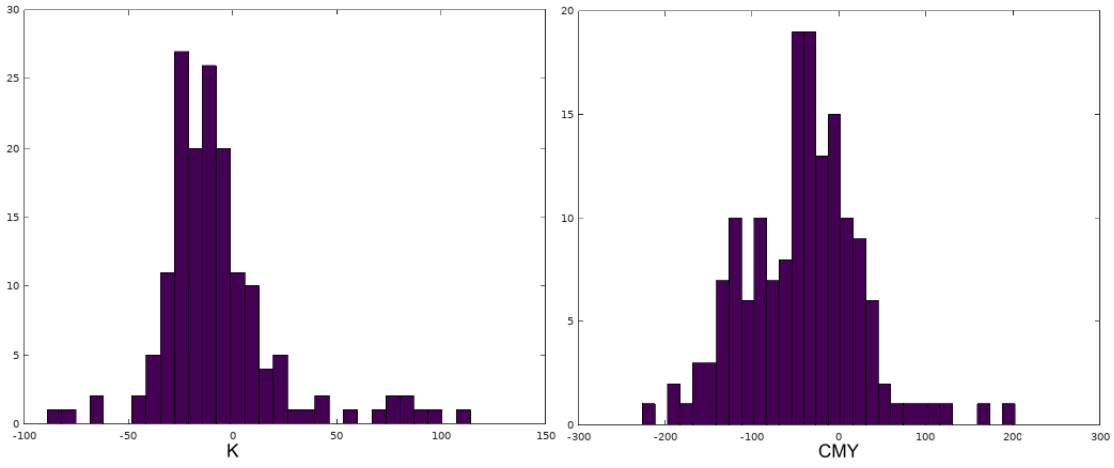


Figure 2: Histograms of toner use

4.2 Multivariate Gaussian distribution

We recall some central statistical concepts. First, the mean or the expected value of a discretely distributed random variable $\underline{x}$ is defined as

$$\mu = \mathrm{E}\underline{x} = \Sigma P(\underline{x} = x_i) \cdot x_i \tag{1}$$

where $P(\underline{x} = x_i)$ is the probability of $\underline{x}$ at x_i. However, our data set now fully defines P, as we have m samples and the probability of each sample is $\frac{1}{m}$. It follows that if we consider each of our samples to be a 2-element vector

$$\mathbf{a}^{(i)} = \begin{bmatrix} a_1^{(i)} \\ a_2^{(i)} \end{bmatrix}, \ \mathbf{a}^{(i)} \in A \tag{2}$$

then for each $a_j^{(i)}, i \in \{1, 2, ..., m\}$ the expected value becomes the average

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} a_j^{(i)} \tag{3}$$

or, when computed simultaneously for both features in Octave

```
mu_A = mean(A,1);
```

Listing 10: Calculating the expected value

with matrix A having been defined in Listing 9.

The second parameter we compute is the covariance matrix Σ (not to be confused with the process of summing). For a vector-valued variable $\mathbf{x} \in \mathrm{R}^n$ the covariance matrix is an n-by-n matrix with each element (j, k) given by

$$\Sigma_{jk} = \mathrm{cov}(x_j, x_k) = \mathrm{E}[x_j x_k] - \mathrm{E}[x_j]\mathrm{E}[x_k] = \mathrm{E}[(x_j - \mu_j)(x_k - \mu_k)] \tag{4}$$

where $x_i, x_j \in \mathbf{x}$. When (4) is expanded for the complete sample set we obtain

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \mathrm{cov}(a_j^{(i)}, a_k^{(i)}) \tag{5}$$

which is easily translated for Octave in Listing 11.

```
Sigma_A = (A.-mu_A)'*(A.-mu_A) ./ size(A,1);
```

Listing 11: Computing the covariance matrix

The covariance matrix is only valid if it is both symmetric, $\Sigma = \Sigma^T$, and positive definite, $x^T \Sigma x > 0,\ \forall x \in \mathrm{R}^n$. In Listing 12 we ensure that these assumptions hold and abort if either of them is invalid.

```
% check for symmetry
if Sigma_A != Sigma_A',
  printf("Covariance matrix is not symmetric! Aborting...\n");
  quit;
end;

% check for positive definiteness, compute eigenvalues
[eig1, eig2] = eig(Sigma_A);
if eig1 < 0 || eig2 < 0,
  printf("Covariance matrix is not positive definite! Aborting...\n");
  quit;
end;
```

Listing 12: Verifying the validity of the covariance matrix

With Σ and $\boldsymbol{\mu}$ we now define the probability density function f for any random sample $\mathbf{x} \in \mathrm{R}^2$ [1]:

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^2 |\Sigma|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})} \tag{6}$$

and if everything is in order, for the cumulative distribution function we have

$$\frac{1}{\sqrt{(2\pi)^2 |\Sigma|}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})} dx_1\ dx_2 = 1 \tag{7}$$

as the basic principle of probability dictates. We use the function **mvncdf**[8] of the Octave statistics package to verify this in Listing 13. Of course, we are computing the actual value in a finitely accurate system, so we inspect the value visually and abort if we consider the value to be too low.

```
% check to see if the cdf is reasonably close to 1
[p, e] = mvncdf([Inf Inf], [-Inf -Inf], mu_A, Sigma_A);
printf("\nCumulative distribution function (-Inf,Inf): %f, estimated error %f\n\n", p, e);
```

Listing 13: Computing the cdf

In the test set the cdf ≈ 0.9995, which we consider sufficient.

To visualize the PDF, we plot the surface graph in Figure 3.

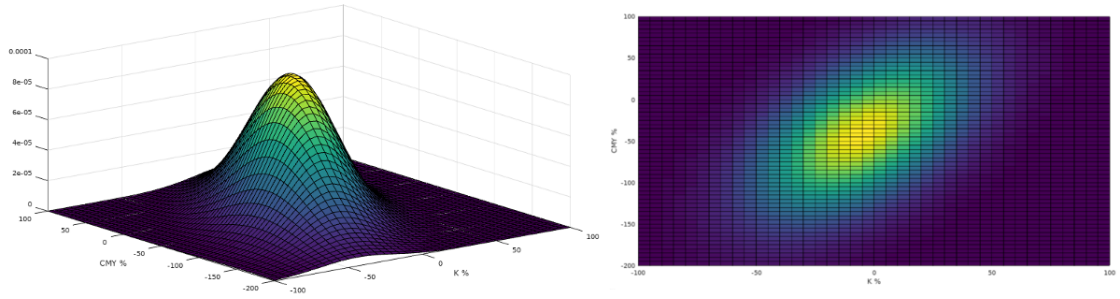


Figure 3: The surface graph of the pdf from side/above

Finally, in Listing 14 we initialize a new matrix C with columns from A and an additional column holding the value of the PDF at each sample. Furthermore, we create a temporary matrix D which holds the rows of C sorted by the column holding the PDF value, and select n samples with the lowest value for the PDF. The choice of n is somewhat arbitrary and will ultimately depend on the choice of how harsh we want the algorithm to be with the sample set. Here, we have chosen the treshold at 30 % of the device count. The value of the PDF at sample number n will become the threshold value ϵ, which will determine if the sample is anomalous (excessive usage) or not. While the portion of 30 % of the whole sample set might seem high, it is worthwhile to know that we will make other adjustments later on.

```
% compute the pdf for each sample
C = [A(:,1) A(:,2) mvnpdf(A, mu_A, Sigma_A)];

% figure out the threshold value epsilon for the pdf;
% below this a sample is initially considered anomalous
% pick epsilon so that about 30 % of samples get tagged
% (we set other additional requirements later!)
D = sortrows(C, 3);
n = round(size(D,1) * 0.3);
epsilon = D(n,3);
clear D;
```

Listing 14: Computing the threshold value

We also determine that every device in the lower-left quadrant of Figure 1 is non-anomalous, as there would be little reason for the algorithm to pick devices which use toner less than the standard amount (see Appendix 1).

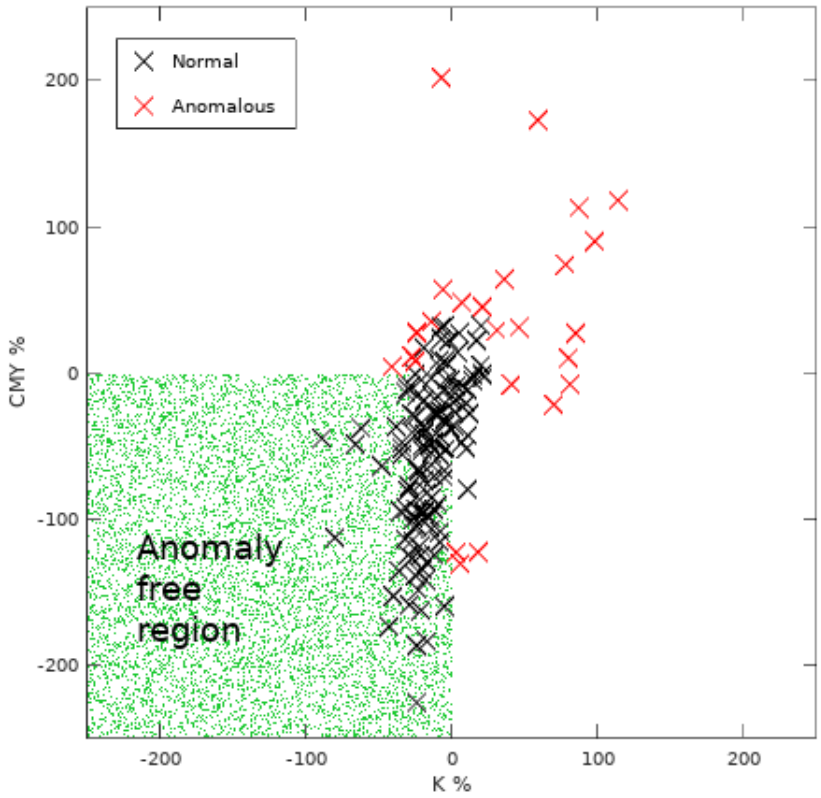


Figure 4: Labeled data

We have now labeled the data. As Figure 4 indicates, this alone is not enough for our sample set to be properly classified.

5 LOGISTIC REGRESSION

Logistic regression is a statistical model (in particular a special case of the regression model) ”where binary response variable is related to a set of explanatory variables, which can be discrete and/or continuous” [9]. That is, it is a type of statistical regression in which the output can take one of two values. Other namings do exist, but due to the prevalence of the term ”logistic regression” in the discipline of machine learning, we too will keep to the same convention. Our modelling is based on the **sigmoid** (or logistic) function [3]

$$f(z) = \frac{1}{1 + e^{-z}} \tag{8}$$

which, when plotted, behaves as shown in Figure 5 ($\lim_{z \to \infty} f(z) = 1$).

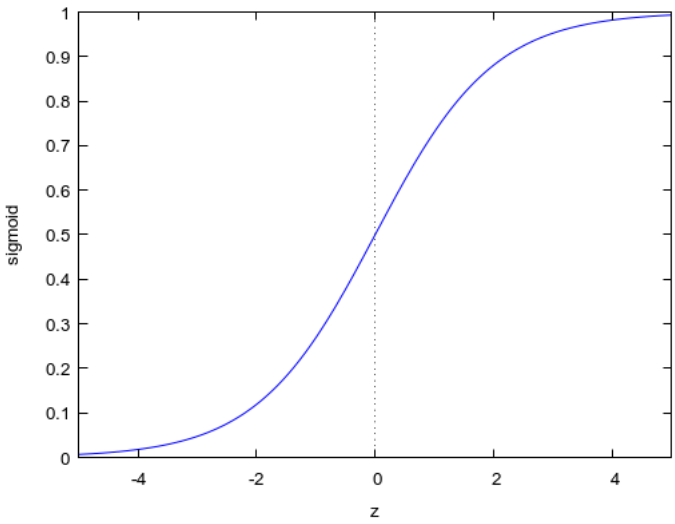


Figure 5: The sigmoid function

The basic principle is to use the sigmoid function to classify samples based on some characteristic z, typically using $z = 0$ as the differentiating boundary. If $f(z) \geq 0.5$, then we classify the sample as 1 (anomalous / excessive), otherwise as 0 (normal).

Another important concept related to the model is that of the **hypothesis** [5], $h_{\boldsymbol{\theta}}(\mathbf{x})$, used for predicting whether a sample is anomalous or not. In the hy-

pothesis the sample $\mathbf{x} \in \mathrm{R}^n$ (with n features) is subjected to a parameter set $\boldsymbol{\theta} \in \mathrm{R}^{n+1}$, with the first element θ_0 being a zero condition, the so-called **bias term**. To make $\mathbf{x}$ compatible with $\boldsymbol{\theta}$ we append unity as the first element of $\mathbf{x}$, resulting as $\mathbf{x}_B = [1\ x_1\ ...\ x_n]^T$ (B for accounted bias). The resulting hypothesis is of the form

$$h_{\boldsymbol{\theta}}(\mathbf{x}_B) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}_B}} \tag{9}$$

which is a composite of $\boldsymbol{\theta}^T \mathbf{x}_B$ and the sigmoid function. The implementation of the hypothesis is straightforward in Octave, as shown in Listing 15 (we assume the parameter X has already been modified for bias).

```
% hypothesis for logistic regression
function H = hypothesis(Theta, X),
  Z = X*Theta';
  O = ones(size(Z,1),1);
  H = O ./ (O + exp(-Z));
end;
```

Listing 15: The hypothesis

This alone is not enough as we do not yet know the actual parameter $\boldsymbol{\theta}$. Instead, we have to *learn* it. The detailed logic, based on the principle of maximum likelihood estimation, can be found in [5]. Here, we simply implement the process.

We first define the **cost function**

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(\mathbf{x}^{(i)} - \mathbf{y}^{(i)} \right)^2 \tag{10}$$

used to compute the cumulative squared error between the predicted and the actual labels. The parameter $\boldsymbol{\theta}$ is then set to some initial value, typically to that of a zero vector, $\boldsymbol{\theta} = \mathbf{0} \in \mathrm{R}^{n+1}$. We next iterate over the process known as **gradient descent** to determine $\boldsymbol{\theta}$ for which the error cost for false predictions is as low as possible.

Gradient descent is defined as

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J = \theta_j - \alpha \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)} \right) x_j^{(i)} \tag{11}$$

for all j simultaneously. That is, we use the same $\boldsymbol{\theta}$ for all j before updating and repeating the process. The parameter α is known as the **learning rate**, which dictates the change of $\boldsymbol{\theta}$ in relation to the derivative of the cost J. An α too large will mean that gradient descent will not be able to converge near the minimal error; too small, and the process will require an excessive number of iterations to converge.

The parameter **y** contains the labels for all the samples **x**. As a result the term $\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)} \right) x_j^{(i)}$ of (11) will be the higher, the further away the hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x})$ is from the actual labels. The closer the hypothesis is to the actual label, the smaller the difference and hence the effect of $x_j^{(i)}$.

The code for gradient descent is shown in Listing 16.

```
% gradient descent for learning values of Theta
function theta = gradient_descent(theta_initial, X, Y, alpha),
  lrounds = 4000; % suitable for the test data in the time of writing
  printf("\nLearning (%d iterations): ", lrounds);
  theta = zeros(size(theta_initial));
  for k=1:4000,
    if(rem(k,100)==0),
      printf(" %d... ", k);
    end;

    theta_temp = theta;
    for j = 1:3,
      summation = 0;
      for i = 1:size(Y,1),
        summation += ((hypothesis(theta_temp, X(i,:)) - Y(i))*X(i,j));
      end;
      theta(j) = theta(j) - alpha .* summation;
    end;
    % uncomment the following if need to see if/how theta is converging
    % printf("%d:\t\t%.5f\t%.5f\t\t%.5f\n", k, theta(1,1), theta(1,2), theta(1,3));
  end;
  printf("Done.\n");
end;
```

Listing 16: Gradient descent

It should be mentioned that as vectors and matrices are first-class citizens in Octave, a vectorized implementation with less for loops would result in a faster execution of gradient descent. Here, the for-looped version is used for clarity.

We are now ready to try learning the parameter θ. We create a new matrix X to hold the actual elements of A as well as the appended column for the bias term (Listing 17).

```
X = [ones(size(A,1),1) A(:,1) A(:,2)];
```

Listing 17: The sample matrix with extra column of ones

We run gradient descent and come up with new values for θ, as in Listing 18.

```
% initial values of the hypothesis parameter theta
theta_initial = [0 0 0];
% learning rate; (note: 0.003 too slow / 0.03 fails to converge)
alpha = 0.01;
% finally get to learning
theta = gradient_descent(theta_initial, X, Y, alpha);
```

Listing 18: Executing gradient descent

The sample set is relabeled using the hypothesis, shown in Figure 6 along with the **decision boundary**, a line separating the two classes in logistic regression. That is, when we pass the boundary, the prediction changes.

The result is expected but clearly unacceptable as it is not possible to meaningfully fit a linear decision boundary to a sample set such as this. We proceed to introduce two additional methods in Listing 19 with which to manipulate the data to a more fitting form.

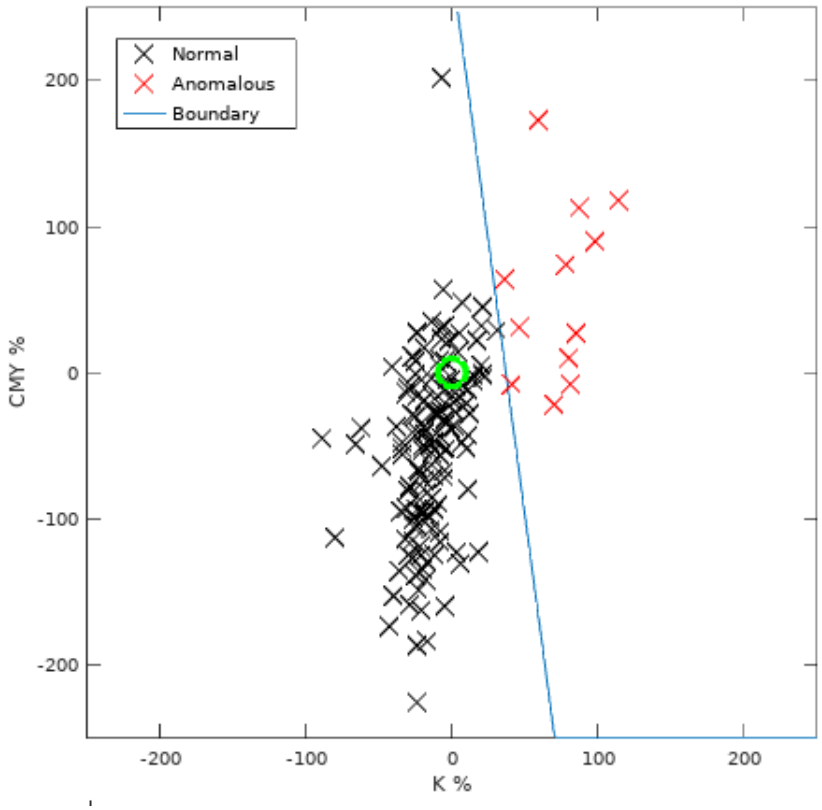


Figure 6: Linear relabeling of the sample set

```
% function to scale the values
function O = scale (X, average, deviation),
 O = (X.-average)./deviation;
end;
% function with which to manipulate the samples to reach a desired detection boundary
function O = manipulate (X),
 O = exp(X);
end;
```

Listing 19: The functions for scaling and manipulating the samples

In addition, we compute the means and the standard deviations for both columns of the sample matrix X, which we then pass on to the **scale** function to normalize the sample set. Without this gradient descent with the manipulated version of X would fail to converge, essentially rendering the learned parameters useless.

```
% compute normalization terms
m2 = mean(X(:,2));
sdev2 = std(X(:,2));
m3 = mean(X(:,3));
sdev3 = std(X(:,3));
% without normalization the learning algorithm will fail to converge
XX = [X(:,1) manipulate(scale(X(:,2), m2, sdev2)) manipulate(scale(X(:,3), m3, sdev3))];
```

Listing 20: Computing the means and the standard deviations

When we run gradient descent again, we end up with a decision boundary that is much more suitable for our purposes. The end result, accompanied by the initial labeling, is shown in Figure 7.

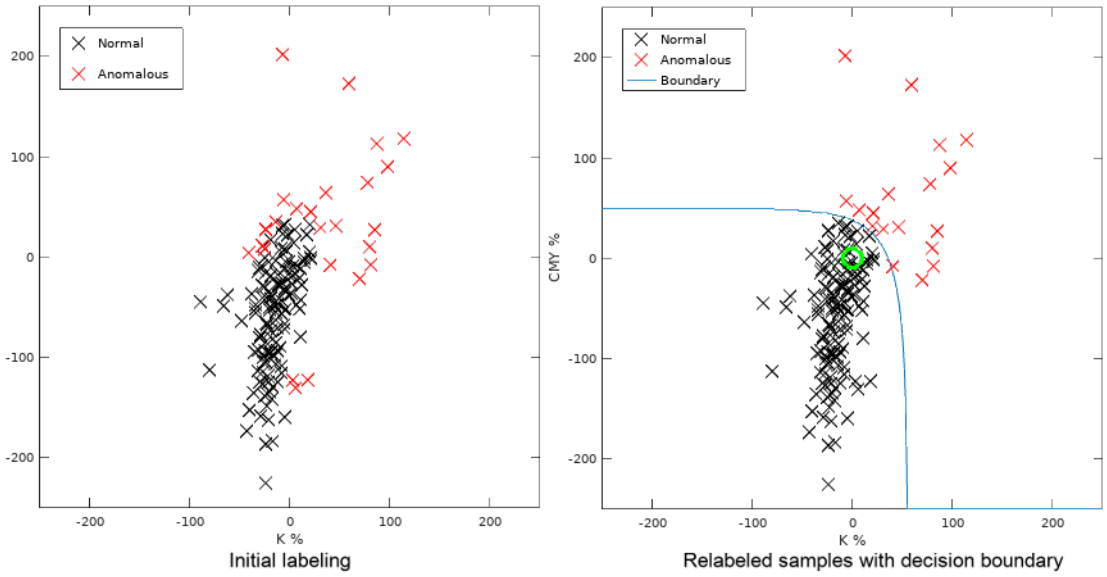


Figure 7: Relabeling after the manipulation of the sample set

To see how much the differences in the initial labeling affect the end result, we go through the process of gradient descent with different percentages of anomalous readings (variable **n** in Listing 14). The results can be examined in Figure 8.

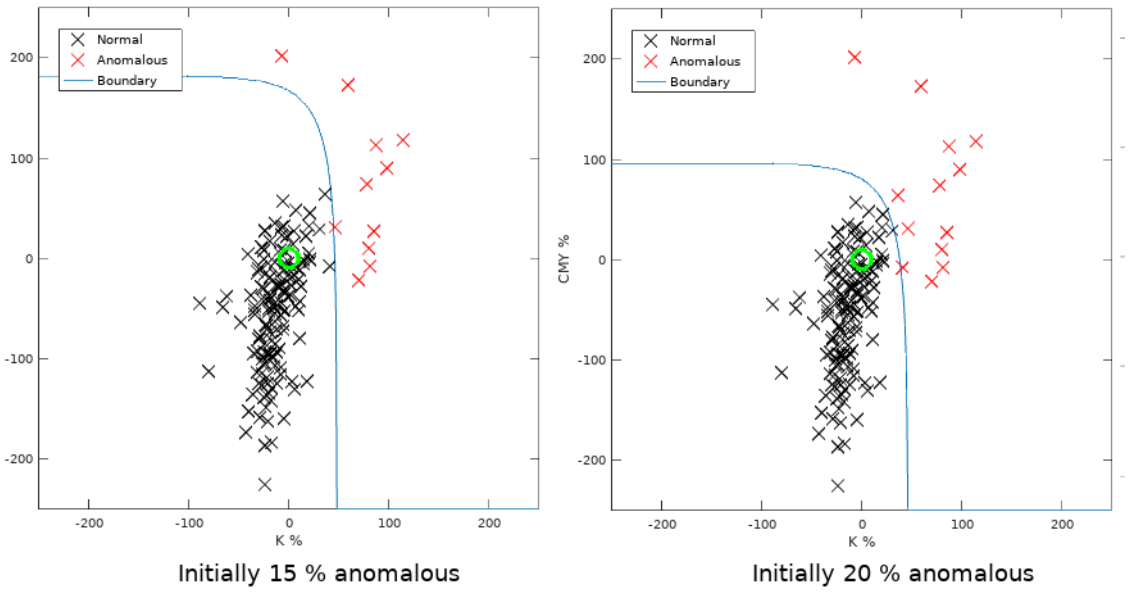


Figure 8: The effect of initial labeling

As seen, not all choices lead to a usable outcome. One way to circumvent this is to use artificial data, that is, include either normal, anomalous or both types of samples of artificial nature in the sample set. However, one has to remember that the more artificial data is included, the less effective the authenthic sample set is. Figure 9 displays the effect of artificial anomalous data, with which we force the decision boundary below the specified limits.

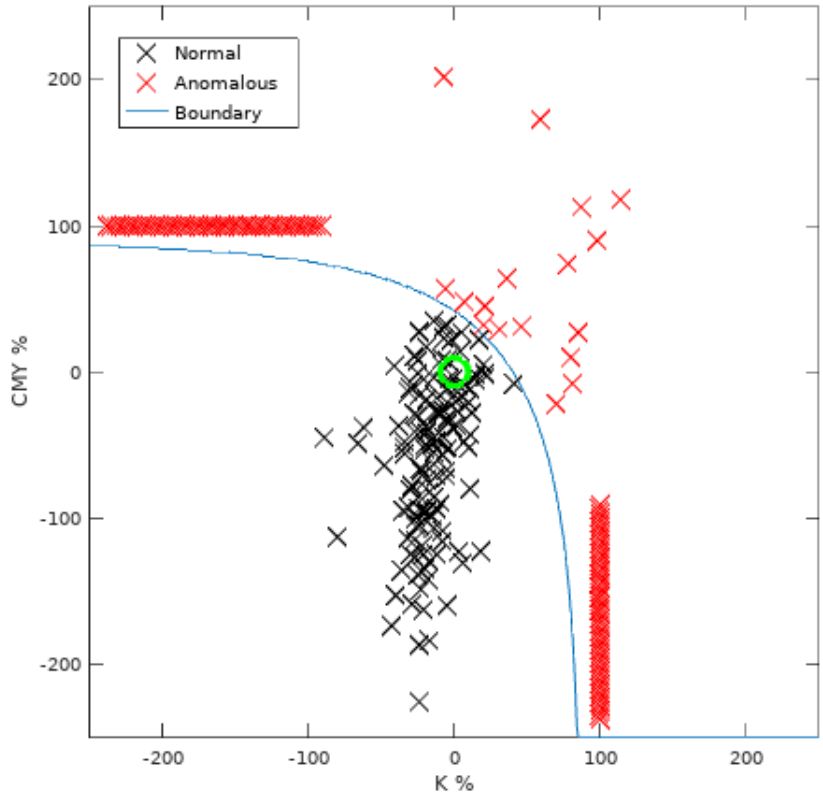


Figure 9: The effect of artificial data (15 % of real samples anomalous)

The last part of the process is testing the hypothesis. In a typical machine learning scenario the sample set is divided into groups. Only one of these groups is used for the actual learning process, while the other groups are reserved for validation and testing purposes. The natural assumption here is that the sampled data has been labeled externally, that is, not labeled in the course of the learning process.

However, as the labeling of our data set is much more ambiguous, such a segregation will only limit the generality of our hypothesis without providing additional benefits. Instead, we choose to create 100 units of randomly generated sample data, each of them acting as a singular addition to the original set. The effect is that every random sample will be considered not to have an effect on the means

and standard deviations already computed in listing 20. Rather, the means and standard deviations will be used to normalize the random test data we have generated. When we do so, we see the desired result in Figure 10.

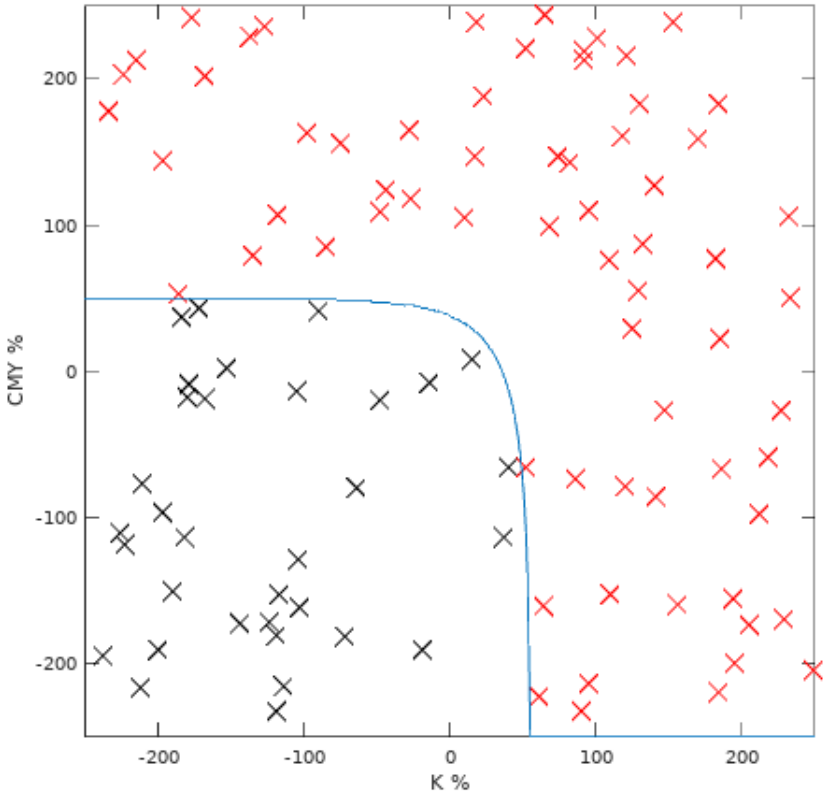


Figure 10: Testing 100 units of random test data

6 IMPLEMENTATION ON A DJANGO SITE

Regardless of the final choices we have made (the percentage of samples marked anomalous, the inclusion of artificial data etc), we now have all the necessary values to implement the relevant parts of the solution in Python using the Django Framework. These include the process for data extraction as well as the hypothesis.

We will, however, omit the actual learning process. This is because whenever the data changes enough to warrant a new learning run, it is impossible to say beforehand if the learning algorithm needs to be modified or run several times. Creating a Django app which can take these considerations into account without impeding the availability of the actual site is usually not worth the time invested.

We begin by creating a model for storing usage readings in Listing 21.

```
class Usage(models.Model):
    device = models.ForeignKey(Device, on_delete=models.CASCADE)
    k_percentage = models.IntegerField(null=True)
    cmy_percentage = models.IntegerField(null=True)
    computable = models.BooleanField(default=True)
    reliable = models.BooleanField(default=True)
    continuous = models.BooleanField(default=False)
    anomalous = models.BooleanField(default=False)
    hidden = models.BooleanField(default=False)
```

Listing 21: The Django model for usage readings

This is only one of the several possible ways of writing the model. In this example the fields of the type *BooleanField* are meant for storing true/false information about each record. Unlike in Chapter 3 we will compute usages for a broader range of devices, because invalid/unreliable information will not affect our hypothesis. Instead, we use the booleans in Listing 21 to monitor the stage of each reading in the following way:

- computable: can the usage reading be computed (no data discrepancies)
- reliable: is the reading reliable (enough data accumulated)
- continuous: does the related device report continuous toner levels
- anomalous: has an anomality been detected
- hidden: in case of anomality, has the reading been hidden from the user

The complete source code for a solution such as this is very similar to that of Appendix 1, and, as said, ultimately defined by the needs of the implementation.

We proceed to define the model for the hypothesis in Listing 22. The first step is to define the fields of the model:

```
class Hypothesis(models.Model):
    theta_bias = models.FloatField()
    theta_k = models.FloatField()
    theta_cmy = models.FloatField()
    k_avg = models.FloatField()
    cmy_avg = models.FloatField()
    k_std = models.FloatField()
    cmy_std = models.FloatField()
```

Listing 22: The Django model for the hypothesis

In the rare case of re-learning the hypothesis parameters these values can simply be inserted via the Django shell, unless there is a real need to construct a Django view or an interface for the learning script to insert new values automatically.

After the introduction of the model we implement (under the Hypothesis class) the actual methods for making predictions (Listing 23, **exp** function from the standard math package).

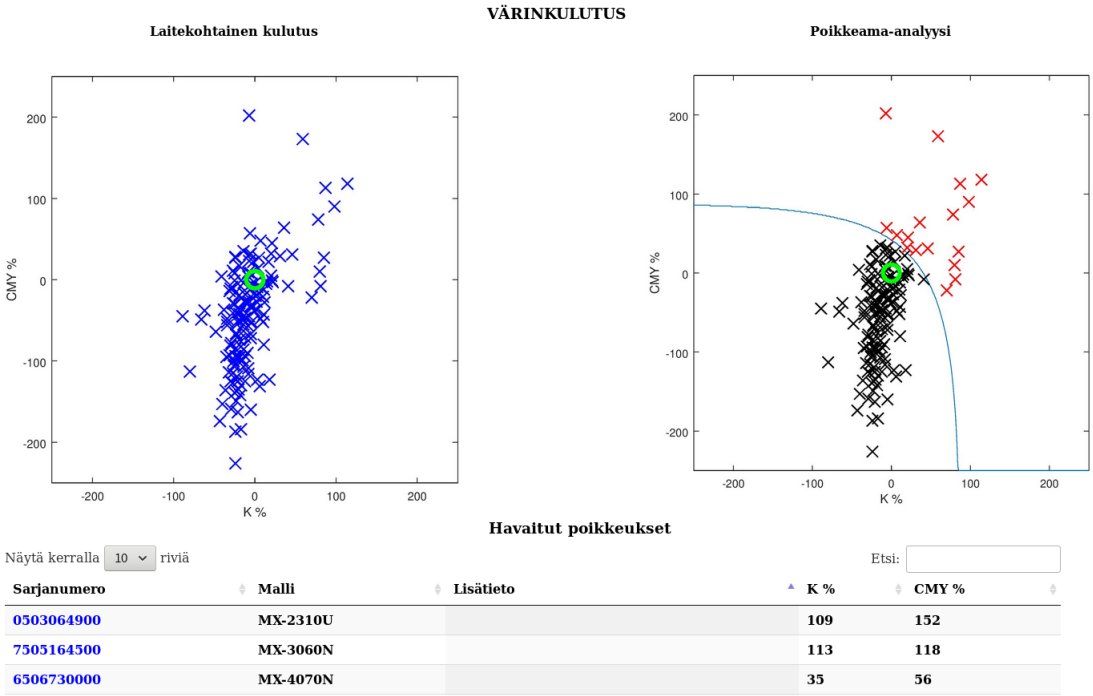
```
@staticmethod
def sigmoid(val):
    return (1/(1+math.exp(-val)))

def predict(self, usages):
    theta = [self.theta_bias, self.theta_k, self.theta_cmy]
    if type(usages) == Usage:
        usages = [usages]
    for u in usages:
        if not u.computable:
            continue
        x1 = math.exp((u.k_percentage - self.k_avg) / self.k_std)
        x2 = math.exp((u.cmy_percentage - self.cmy_avg) / self.cmy_std)
        # compute the "simulated" dot product of usage and theta
        dot_product = theta[0] + theta[1] * x1 + theta[2] * x2
        sigmoid_result = Hypothesis.sigmoid(dot_product)
        u.anomalous = True if sigmoid_result >= 0.5 else False
        u.save()
```

Listing 23: Computing the hypothesis in Python

The **predict** function can be used to predict the anomality of both old usage readings and new ones not present at the time the parameters were learned. The only limitation is that if the device fleet grows or transforms over time, one needs to evaluate when to re-learn the parameters.

In addition, some other implementation specific decisions are needed. For example, one needs to decide how to present anomalous usages to the user, if and what plotting tools are needed for the visualization of the data etc. These, however, have more to do with UI design than with the actual topic of this thesis. For demonstration, one possible view is presented in Picture 4.

To the left, a general usage distribution is shown; to the right, the samples marked anomalous and the decision boundary. Below these are some examples of devices detected to have excessive toner usage.



Picture 4: The usage analysis view in PCC

7 CONCLUSIONS AND REMARKS

The increasing performance of personal computers and the widely available tools and libraries have made it easier than ever to apply machine learning principles and procedures in solving complex problems without precise answers. The objective of this thesis was to utilize these tools in order to provide a detailed toner usage analysis of a printer fleet consisting of hundreds of devices and offer the possibility to automatically pinpoint devices with excessive toner use.

The analytical methods devised in this thesis consist of three central phases:

1. transforming the device-specific readings into an aggregated data set
2. labeling the data using the principles of multivariate Gaussian distributions
3. classifying the data using the principles of logistic regression

The results are somewhat close to what was expected. They demonstrate that such an analysis is indeed possible and that it offers a way for the fleet manager

to monitor the fleet and toner consumption in a sensible way. In part this is possible because of the accuracy of the ISO/IEC standard 19798:2017, at least with respect to the sample set used in this thesis. The distribution of the devices is the most dense roughly around the standard point, and the devices that differ are distributed so that handling them with the methods of Gaussian distribution provides viable results.

On the other hand the results display a certain level of ambiguity that can be considered inseparable in nature. This stems from the fact that the classification between normal and anomalous toner usage is always a matter of subjective choice. In the end it is up to the fleet manager to decide how to implement the solution to offer an outcome best suited for a particular organization.

As to the further development of these methods, the most obvious road is that of improved usability and dynamics. In the current form the learning requires manual intervention to happen, and this is adequate as long as there are no sudden changes in the fleet. However, if the fleet were to undergo such changes constantly, a different kind of approach would be needed. The most simple one would be to schedule periodic reruns of the Octave script in Appendix 2, the results of which could then be stored, perhaps as simply as in a plain-text file. The Django implementation would then read this file and update the parameters of the hypothesis accordingly. The drawback of this is that automatic runs of the learning phase will either be subject to disruptions and errors in case the sample set does not fit the default algorithm anymore. Or, additional effort would be required to make sure this does not happen.

In the present state the methods in this thesis offer a middle ground solution between accuracy, automation and effort, which is suitable for many independent local MFP dealers.

8 REFERENCES

[1] Do, Chuong B. 2008. The Multivariate Gaussian Distribution. Stanford University. Referenced 13.12.2017 http://cs229.stanford.edu/section/gaussians.pdf

[2] GNU Project. About GNU Octave. Referenced 12.12.2017. https://www.gnu.org/software/octave/about.html

[3] Kleinbaum, David G. & Klein, Mitchel. 2010. Logistic Regression. A Self-Learning Text. 3rd edition. Springer.

[4] ISO/IEC. 19798:2017. 2017.

[5] Ng, Andrew. Machine learning: Lecture6.pdf. Stanford University. Referenced 17.12.2017 https://www.coursera.org/learn/machine-learning

[6] Ng, Andrew. Machine learning: Lecture15.pdf. Stanford University. Referenced 17.12.2017 https://www.coursera.org/learn/machine-learning

[7] Onken, Arno et al. 2016. statistics v1.3.0. Referenced 12.12.2017. https://octave.sourceforge.io/statistics/

[8] Onken, Arno et al. Function reference: mvncdf. Referenced 17.12.2017. https://octave.sourceforge.io/statistics/function/mvncdf.html

[9] The Pennsylvania State University. 2017. Lesson 6: Logistic Regression. Referenced 22.12.2017. https://onlinecourses.science.psu.edu/stat504/node/149

APPENDIX 1. The data extraction script in Python

```
import sqlite3

# The format of each entry:
# [SN, BW#, COL#, K%, C%, M%, Y%, K#, C#, M#, Y#]
# Indexes, if need to change the format
I_SN = 0
I_MOD = 1
I_BW = 2
I_COL = 3
# Toner levels
I_KI = 4
I_CI = 5
I_MI = 6
I_YI = 7
level_index = [I_KI, I_CI, I_MI, I_YI]
# Installed cartridge counts
I_Kc = 8
I_Cc = 9
I_Mc = 10
I_Yc = 11
counts_index =[I_Kc, I_Cc, I_Mc, I_Yc]

def levels_to_ints(stat_line):
    stat_line = list(stat_line)

    # keep track that the initial levels were intervals or continuous
    stat_line.append('interval' if '-' in stat_line[I_KI] else 'continuous')

    for i in level_index:
        # older machines report toner levels in intervals 75-100%, 50-75% etc
        # we take the lower limit and add half (rounded down) to approximate
        if '-' in stat_line[i]:
            stat_line[i] = int(stat_line[i][0:stat_line[i].index('-')])+12
            stat_line.append('interval')
        else:
            stat_line[i] = int(stat_line[i].strip('%'))

    return stat_line


db = sqlite3.connect('../db.sqlite3')
cur = db.cursor()

cur.execute('''
  SELECT serial_number, model, total_bw, total_col,
  k_level, c_level, m_level, y_level,
  k_installed, c_installed, m_installed, y_installed
  FROM pcc_total INNER JOIN pcc_device
  ON pcc_total.device_id = pcc_device.id WHERE c_installed > -1
  ORDER BY pcc_device.serial_number, pcc_total.date_sent
  ''')

# list statistics of the devices, device per entry; this time the values
# indicate change in totals: [SN, BW#, COL#, K%, C%, M%, Y%, K#, C#, M#, Y#]
```

```
device_stats = []

# for the computational phase; we exclude devices with too few entries
entries_per_device = 0
MIN_ENTRY_COUNT = 2

# minimal levels for toner changes for accepted values
MIN_CONT_CHANGE = 5
MIN_INTERV_CHANGE = 125

# set some values that need to exist for comparisons
new_entry = []
serial_number = ''
stats_valid = True  # to check if the stats are valid, see below

# yield groups, for different devices using different toners with diff. yields
# var = standard pages (5 % coverage) / 1 % of toner
BK23 = 180
COL23 = 100
BK36 = 240
COL36 = 150
BK51 = 400
COL51 = 180
BK60 = 400
COL60 = 240

yields = {
        'MX-2310U': [BK23, COL23],
        'MX-2314N': [BK23, COL23],
        'MX-2614N': [BK23, COL23],
        'MX-3111N': [BK23, COL23],
        'MX-3114N': [BK23, COL23],
        'MX-2610N': [BK36, COL36],
        'MX-2640N': [BK36, COL36],

        'MX-3110N': [BK36, COL36],
        'MX-3610N': [BK36, COL36],
        'MX-3640N': [BK36, COL36],

        'MX-4112N': [BK51, COL51],
        'MX-4140N': [BK51, COL51],
        'MX-4141N': [BK51, COL51],
        'MX-5112N': [BK51, COL51],
        'MX-5140N': [BK51, COL51],
        'MX-5141N': [BK51, COL51],

        'MX-3050N': [BK60, COL60],
        'MX-3060N': [BK60, COL60],
        'MX-3070N': [BK60, COL60],

        'MX-3550N': [BK60, COL60],
        'MX-3560N': [BK60, COL60],
        'MX-3570N': [BK60, COL60],

        'MX-4050N': [BK60, COL60],
        'MX-4060N': [BK60, COL60],
        'MX-4070N': [BK60, COL60],
```

```
112
113         'MX-5050N': [BK60, COL60],
114         'MX-5060N': [BK60, COL60],
115         'MX-5070N': [BK60, COL60],
116
117         'MX-6040N': [BK60, COL60],
118         'MX-6050N': [BK60, COL60],
119         'MX-6070N': [BK60, COL60],
120         }
121
122 while True:
123     try:
124         previous_entry = new_entry
125         new_entry = levels_to_ints(cur.fetchone())
126     except TypeError:
127         new_entry = ['end']
128
129     # if serial changes, new_entry is for the next device in the query
130     if serial_number != new_entry[I_SN]:
131         if entries_per_device >= MIN_ENTRY_COUNT:
132
133             # previous_entry is the last entry for the device now analyzed
134             # change in the number of printed pages
135             bw = previous_entry[I_BW] - bw_first
136             col = previous_entry[I_COL] - col_first
137
138             # change in the installed number of cartridges
139             k_crtg = previous_entry[I_Kc] - k_ctrg_first
140             c_crtg = previous_entry[I_Cc] - c_ctrg_first
141             m_crtg = previous_entry[I_Mc] - m_ctrg_first
142             y_crtg = previous_entry[I_Yc] - y_ctrg_first
143
144             # change in toner levels
145             k_level = k_level_first + 100 * k_crtg - previous_entry[I_Kl]
146             c_level = c_level_first + 100 * c_crtg - previous_entry[I_Cl]
147             m_level = m_level_first + 100 * m_crtg - previous_entry[I_Ml]
148             y_level = y_level_first + 100 * y_crtg - previous_entry[I_Yl]
149
150
151             # if the change in the number of pages is small, we should probably
152             # exclude the device for the time being, as we have not enough data
153             if bw + col > 5000 and stats_valid:
154
155                 # due to the approximation in limits_to_ints, we can not
156                 # include interval monitored devices with low level changes
157                 if previous_entry[-1] == 'interval':
158                     if (k_level < MIN_INTERV_CHANGE
159                             or c_level < MIN_INTERV_CHANGE
160                             or m_level < MIN_INTERV_CHANGE
161                             or y_level < MIN_INTERV_CHANGE):
162                         stats_valid = False
163
164                 # also in devices with continuous levels we should see at
165                 # least some change before considering the values safe
166                 elif previous_entry[-1] == 'continuous':
167                     if (k_level < MIN_CONT_CHANGE
168                             or c_level < MIN_CONT_CHANGE
```

```
                            or m_level < MIN_CONT_CHANGE
                            or y_level < MIN_CONT_CHANGE):
                        stats_valid = False

                if stats_valid:
                    try:
                        bk_yield = yields[previous_entry[I_MOD]][0]
                        col_yield = yields[previous_entry[I_MOD]][1]

                        # consumed toner in standard pages (5 % coverage)
                        std_k = k_level * bk_yield
                        std_c = c_level * col_yield
                        std_m = m_level * col_yield
                        std_y = y_level * col_yield

                        # a dict later exported into a csv file
                        stat_dict = {
                                'bw': str(bw),
                                'col': str(col),
                                'std_k': str(std_k),
                                'std_c': str(std_c),
                                'std_m': str(std_m),
                                'std_y': str(std_y),
                                }

                        device_stats.append(stat_dict)
                    except KeyError:
                        # no yield defined for the (too old) model , drop it
                        pass
                    except ZeroDivisionError:
                        # one of the yields was 0?
                        pass

            if new_entry[I_SN] != 'end':
                stats_valid = True

                # begin to analyze the next device
                entries_per_device = 1
                serial_number = new_entry[I_SN]

                # the initial numbers of printed pages
                bw_first = new_entry[I_BW]
                col_first = new_entry[I_COL]

                # initial % levels of toners
                k_level_first = new_entry[I_Kl]
                c_level_first = new_entry[I_Cl]
                m_level_first = new_entry[I_Ml]
                y_level_first = new_entry[I_Yl]

                # initial number of installed cartridges
                k_ctrg_first = new_entry[I_Kc]
                c_ctrg_first = new_entry[I_Cc]
                m_ctrg_first = new_entry[I_Mc]
                y_ctrg_first = new_entry[I_Yc]

            else:
```

```
226             break
227
228     else:
229         # if the toner levels have increased but the number of installed
230         # cartridges has not, there's bound to be problems -> ignore device
231         for i in range(0,4):
232             if (new_entry[level_index[i]] > previous_entry[level_index[i]] and
233                     previous_entry[counts_index[i]] >=
234                     new_entry[counts_index[i]]):
235                 stats_valid = False
236
237         entries_per_device += 1
238         previous_entry = new_entry
239
240 with open('stats.csv', 'w', encoding='utf-8') as file:
241     for s in device_stats:
242         file.write(s['bw'] + ',' + s['col'] + ',' + s['std_k'] + ',' +
243                 s['std_c'] + ',' + s['std_m'] + ',' + s['std_y'] + '\n')
244
245 db.close()
246
```

APPENDIX 2. The data analysis script in Octave

```
% Anomaly / toner overuse detection, an extension for Python Copy Counter
% Timo Jokela, 2017
% This script is for prototyping and hypothesis fitting purposes;
% monitored execution only!

pkg load statistics;
warning("off");

% legend position and size
rect = [0.33, 0.79, 0.1, 0.1];

%%%%%%%%%%%%%%%%%%%%%%%
%Function definitions%
%%%%%%%%%%%%%%%%%%%%%%%

% function to scale the values
function O = scale (X, average, deviation),
  O = (X.-average)./deviation;
end;

% function with which to manipulate the samples to reach a desired detection boundary
function O = manipulate (X),
  O = exp(X);
end;

% hypothesis for logistic regression
function H = hypothesis(theta, X),
  Z = X*theta';
  O = ones(size(Z,1),1);
  H = O ./ (O + exp(-Z));
end;

% gradient descent for learning values of theta
function theta = gradient_descent(theta_initial, X, Y, alpha),
  lrounds = 4000; % suitable for the test data in the time of writing
  printf("\nLearning (%d iterations): ", lrounds);
  theta = zeros(size(theta_initial));
  for k=1:4000,
    if(rem(k,100)==0),
      printf(" %d... ", k);
    end;

    theta_temp = theta;
    for j = 1:3,
      summation = 0;
      for i = 1:size(Y,1),
        summation += ((hypothesis(theta_temp, X(i,:)) - Y(i))*X(i,j));
      end;
      theta(j) = theta(j) - alpha .* summation;
    end;
    % uncomment the following if need to see ig/how theta is converging
    % printf("%d:\t\t%.5f\t%.5f\t\t%.5f\n", k, theta(1,1), theta(1,2), theta(1,3));
  end;
  printf("Done.\n");
```

```
end;

%%%%%%%%end function definitions%%%%%%%%%
printf("\nBegin the learning process.\n");

% read data csv => matrix M
M = csvread('stats.csv');

% matrix A for the initial inspection of the data
A = zeros(size(M,1), 2);
for i = 1:size(M,1),
  % average use of CMY toners
  cmy_std_avg = mean(M(i,4:6),2);

  % ratio of standard pages (5% coverage) to actually printed pages
  k_std_to_real = M(i,3) / (M(i,1) + M(i,2));
  cmy_std_to_real = cmy_std_avg / M(i,2);

  % ratio of actually printed pages to standard pages
  k_real_to_std = (M(i,1) + M(i,2)) / M(i,3);
  cmy_real_to_std = M(i,2) / cmy_std_avg;

  % pick wichever is larger, and adjust so that std>real
  % equals moderate toner use (mark negative)
  % and real>std equals excessive toner use (mark positive),
  % also the ratio 1 equals to standard
  % so shift to zero (+/- 1)
  if k_std_to_real > k_real_to_std,
    k = k_std_to_real - 1;
  else,
    k = -k_real_to_std + 1;
  end;

  if cmy_std_to_real > cmy_real_to_std,
    cmy = cmy_std_to_real - 1;
  else,
    cmy = -cmy_real_to_std + 1;
  end;

  % finally convert to percentages
  A(i,1) = round(k*100);
  A(i,2) = round(cmy*100);

end;

% plot devices on a K/CMY chart
printf("\nPlotting K/CMY chart...\n");
figure(1);
plot(A(:,1), A(:,2), 'bx', 'MarkerSize', 10, 'LineWidth', 1.0);
hold on;
axis([-250, 250, -250, 250], "square");
xlabel("K %");
ylabel("CMY %");
plot(0,0, 'go', 'MarkerSize', 15, 'LineWidth', 3.5);
hold on;
h = legend("Device", "Standard use");
ch = get(h, "Children");
```

```
112 set(ch(1), "Color", [0 1 0]);
113 set(ch(2), "Color", [0 0 1]);
114 set(h, "Position", rect);
115 printf('———Press any key to proceed———\n');
116 pause;
117
118 % histograms of K and CMY distributions
119 printf("\nPlotting histograms...\n");
120 close all;
121 figure(1);
122 hist(A(:,1), 30);
123 figure(2);
124 hist(A(:,2), 30);
125 printf('———Press any key to proceed———\n');
126 pause;
127
128 % compute the mean of the distributions and the covariance matrix
129 printf("\nComputing the parameters of the Gaussian distribution...\n");
130 mu_A = mean(A,1);
131 Sigma_A = (A.-mu_A)'*(A.-mu_A) ./ size(A,1);
132
133 printf("\nGaussian parameters of the sample matrix A:\n\nAverages:\n");
134 disp(mu_A);
135 printf("\nCovariance matrix:\n");
136 Sigma_A
137
138 % check for symmetry
139 if Sigma_A != Sigma_A',
140   printf("Covariance matrix is not symmetric! Aborting...\n");
141   quit;
142 end;
143
144 % check for positive definiteness, compute eigenvalues
145 [eig1, eig2] = eig(Sigma_A);
146 if eig1 < 0 || eig2 < 0,
147   printf("Covariance matrix is not positive definite! Aborting...\n");
148   quit;
149 end;
150
151 % check to see if the cdf is reasonably close to 1
152 [p, e] = mvncdf([Inf Inf], [-Inf -Inf], mu_A, Sigma_A);
153 printf("\nCumulative distribution function (-Inf,Inf): %f, estimated error %f\n\n", p, e);
154
155 % a surface plot for visual verification of the situation
156 printf("Plotting surface graph of the pdf...\n");
157 close all;
158 figure(1);
159 x1 = -100:5:100;
160 x2 = -200:5:100;
161 [X1, X2] = meshgrid(x1, x2);
162 S = mvnpdf([X1(:) X2(:)], mu_A, Sigma_A);
163 S = reshape(S, length(x2), length(x1));
164 surf(x1, x2, S);
165 hold on;
166 plot(0,0, 'ro');
167 xlabel("K %");
168 ylabel("CMY %");
```

```
169
170 % compute the pdf for each sample
171 C = [A(:,1) A(:,2) mvnpdf(A, mu_A, Sigma_A)];
172
173 % figure out the threshold value epsilon for the pdf;
174 % below this a sample is initially considered anomalous
175 % pick epsilon so that about 30 % of samples get tagged
176 % (we set other additional requirements later!)
177 D = sortrows(C, 3);
178 n = round(size(D,1) * 0.30);
179 epsilon = D(n,3);
180 clear D;
181
182 printf("Threshold of the pdf below which the sample is anomalous:\n");
183 epsilon
184 printf('\n------Press any key to proceed------\n');
185 pause;
186 close all;
187
188 % mark the sample as normal if pdf over theshold or both K and CMY uses below standard,
189 % otherwise anomalous
190 printf("\nMarking the samples...\n");
191 Y = zeros(size(C,1),1);
192 h = zeros(size(C,1),1);
193 figure(1);
194 for i = 1:size(A,1),
195   if C(i,3) > epsilon || (C(i,1) < 0 && C(i,2) < 0),
196     Y(i) = 0;
197     plot(A(i,1), A(i,2), 'kx', 'MarkerSize', 10, 'LineWidth', 1.0);
198     hold on;
199   else,
200     plot(A(i,1), A(i,2), 'rx', 'MarkerSize', 10, 'LineWidth', 1.0);
201     hold on;
202     Y(i) = 1;
203   end;
204 end;
205 xlabel("K %");
206 ylabel("CMY %");
207 axis([-250, 250, -250, 250], "square");
208 h = legend("Normal", "Anomalous");
209 ch = get(h, "Children");
210 set(ch(1), "Color", [1 0 0]);
211 set(ch(2), "Color", [0 0 0]);
212 set(h, "Position", rect);
213 hold on;
214 printf("\nDone.\n");
215 printf('\n------Press any key to proceed------\n');
216 pause;
217
218 % add column of ones for the bias in theta
219 X = [ones(size(A,1),1) A(:,1) A(:,2)];
220
221 % generate artificial normal/anomalous samples to account for the lack of data
222 %X0 = ones(100,3);
223 %Y0 = zeros(100,1);
224 %X1 = ones(100,3);
225 %Y1 = ones(100,1);
```

```
%% an excess of 50% normal, an excess of 100% anomalous
%for i = 1:3:150,
% X0(floor(i/3)+rem(i,3),:) = [1 -i-90 50];
% X0(floor(i/3)+rem(i,3)+50,:) = [1 50 -i-90];
% X1(floor(i/3)+rem(i,3),:) = [1 -i-90 100];
% X1(floor(i/3)+rem(i,3)+50,:) = [1 100 -i-90];
%end;
% add to the end of the sample set and the label set
%X = [X; X0; X1];
%Y = [Y; Y0; Y1];
%X = [X; X0];
%Y = [Y; Y0];
%X = [X; X1];
%Y = [Y; Y1];

% compute normalization terms
m2 = mean(X(:,2));
sdev2 = std(X(:,2));
m3 = mean(X(:,3));
sdev3 = std(X(:,3));
printf("\nValues for normalization:\n\nK average: ...
  \t%f\t\tK standard deviation:\t%f\n", m2, sdev2);
printf("\nCMY average:\t%f\t\tCMY standard deviation:\t%f\n", m3, sdev3);

% without normalization the learning algorithm will fail to converge
XX = [X(:,1) manipulate(scale(X(:,2), m2, sdev2)) manipulate(scale(X(:,3), m3, sdev3))];

% initial values of the hypothesis parameter theta
theta_initial = [0 0 0];
% learning rate; (note: 0.003 too slow / 0.03 fails to converge)
alpha = 0.01;
% finally get to learning
theta = gradient_descent(theta_initial, XX, Y, alpha);
printf("\nLearned values of theta:\n");
disp(theta);

% predictions of the sample set
YY = (hypothesis(theta, XX) >= 0.5);

% plotting the decision boundary; go over every x in (-250,250)
% and see at which y the prediction changes
printf("\nPlotting the decision boundary...\n");
N = zeros(501,1);
first = last = -999;
for a = -250:1:250,
  for b = -250:1:250,
    % remember to normalize the boundary plotting values too for theta to hold
    c = manipulate(scale(a, m2, sdev2));
    d = manipulate(scale(b, m3, sdev3));
    L = [1 c d];
    res = hypothesis(theta, L);
    if res >= 0.5,
      N(251+a) = b;
      if first == -999,
        first = a;
      else
        last = a;
```

```
283       end;
284       break;
285     end;
286   end;
287 end;
288
289 figure(1);
290 hold on;
291 xlabel("K %");
292 ylabel("CMY %");
293 axis([-250, 250, -250, 250], "square");
294 plot((first:1:last),N(251+first:251+last,:));
295 hold on;
296 for i = 1:size(X,1),
297   if  YY(i) == 0,
298     plot(X(i,2), X(i,3), 'kx', 'MarkerSize', 10, 'LineWidth', 1.0);
299     hold on;
300   else,
301     plot(X(i,2), X(i,3), 'rx', 'MarkerSize', 10, 'LineWidth', 1.0);
302     hold on;
303   end;
304   if i == size(A,1),
305     printf("\nPlotting actual samples finished. Press any key to plot artificial data.");
306     pause;
307     printf("Plotting... ");
308   end;
309 plot(0,0, 'go', 'MarkerSize', 15, 'LineWidth', 3.5);
310 h = legend("Normal", "Anomalous", "Boundary");
311 ch = get(h, "Children");
312 % Legend for boundary displays dot instead of line, needs fixing
313 set(ch(1), "Color", [0 0 1], "Marker", ".");
314 set(ch(2), "Color", [1 0 0]);
315 set(ch(3), "Color", [0 0 0]);
316 set(h, "Position", rect);
317 hold on;
318 end;
319 printf("\nDone.\n");
320 printf('\n----Press any key to proceed----\n');
321
322 pause;
323 close all;
324 printf("\nGenerating %d samples of random test data...", n);
325 % random test samples
326 n = 100;
327 T = ones(n,3);
328 TY = zeros(n,1);
329 for i = 1:n,
330   k = round(rand()*250) * (-1)^(round(rand()));
331   cmy = round(rand()*250) * (-1)^(round(rand()));
332   T(i,:) = [1 k cmy];
333 end;
334
335 % normalize the test samples;
336 % we will treat each sample as a singular addition to the original sample set,
337 % that is, we do not compute averages/stds for the test set,
338 % as that would not fit the hypothesis
339 TX = [T(:,1) manipulate(scale(T(:,2), m2, sdev2)) manipulate(scale(T(:,3), m3, sdev3))];
```

```
%TX = [T(:,1) exp((T(:,2).-m2)./sdev2) exp((T(:,3).-m3)./sdev3)];
TY = (hypothesis(theta, TX) >= 0.5);

figure(1);
for i = 1:size(T,1),
  if  TY(i) == 0,
    plot(T(i,2), T(i,3), 'kx', 'MarkerSize', 10, 'LineWidth', 1.0);
    hold on;
  else,
    plot(T(i,2), T(i,3), 'rx', 'MarkerSize', 10, 'LineWidth', 1.0);
    hold on;
  end;
end;
plot((first:1:last),N(251+first:251+last,:));
hold on;
xlabel("K %");
ylabel("CMY %");
axis([-250, 250, -250, 250], "square");
printf("\n\nProcess finished.\nPress any key to quit.\n\n");
pause;

```