

IKT-407 Project 3
in
Web Content Mining

Arild Finne and Erik Trædal

November 19, 2004

Executive Summary

Most web pages are badly programmed, and do not follow the open defined standards, and thus the availability is reduced. Text and mathematical formulas are a good example of wrong use of images that leads to reduced availability, especially for blind people. The better ways to represent text is of course pure text in HTML format, and formulas can be represented brilliantly in the open standard MathML, defined by W3C.

This project lead to the making of a web crawler which retrieves and categorise images on web pages. The images are categorised in *real photos*, *graphics* and *text* (which include formulas).

To find the right approaches and algorithms, a test program was written. This test program tested the algorithms on 20 images from each of the categories. Several algorithms showed big difference on the different categories.

The best algorithm turned out to be one based on Benford's law ([1]). Natural photos follow the law (which claims that most things start with the number 1), better then graphics. The real photos are sorted out very good, but the algorithm is unfortunately very slow.

The Entropy gets almost the same results as the Benford algorithm, only a bit less exact, and is very quick. Therefore this algorithm is primarily used, and in cases of doubt the benford algorithm is used.

A pixel search algorithm is also used. This algorithm is called line-scan, and separates the text category very clearly from the rest. In addition, a pattern search algorithm based on theory from Poisson is used. This algorithm is not especially good, so it does not have too much to say in the final decision of category.

Design elements like borders, lines, small icons and margins are categorised as graphic based on its size. Unusually big images that may slow down the categorising process significantly are also excluded, based on its size. With other words, no algorithms are run on these objects, so time is saved.

The final prototype has a very good hit rate, and fulfils the objectives completely. During this project, the authors have realized that an 100% exact categorisation is close to impossible due to the diversity of the images. The images in the different categories blend into each other.

This solution can be used to reveal the bad use of graphics at web pages. It might also be used as a plug-in in browsers to give additional image information for those who cannot see them.

Contents

- 1 Introduction 4**
 - 1.1 Motivation 4
 - 1.2 The objectives of this project 5
 - 1.3 The structure of this report 6

- 2 Categorising approaches and algorithms 7**
 - 2.1 Categorising approaches 7
 - 2.1.1 Format 7
 - 2.1.2 Size 8
 - 2.1.3 Histogram 8
 - 2.1.4 Image conversion 9
 - 2.1.5 Crop image into sub image 9
 - 2.2 Algorithms 9
 - 2.2.1 Standard deviation 9
 - 2.2.2 Entropy 10
 - 2.2.3 Pixel search 10
 - 2.2.4 Pattern recognition with Poisson 11
 - 2.2.5 Benford's law 12

- 3 The testbed 13**
 - 3.1 Test method 13
 - 3.2 Test results 13
 - 3.2.1 Entropy 14
 - 3.2.2 Linescan 14
 - 3.2.3 Poisson 15
 - 3.2.4 Benford 16
 - 3.2.5 Standard deviation 17

- 4 The final implementation 18**
 - 4.1 The crawler with its web mining part 18
 - 4.2 PictureContainer 19

CONTENTS

4.3 The categoriser	20
5 Summary	21

List of Figures

1.1	MathML shown in Mozilla	5
2.1	Poisson results with different box sizes. The box vary from small to big along the X-axis. One side of the square boxes is five times bigger then the X-value. The dotted line is the <i>mean value</i> and the solid line is the <i>variance</i>	12
3.1	Entropy test results	14
3.2	Line-scan test results. Real and realistic image have an equal graph.	15
3.3	Poisson test results	15
3.4	Benford test results	16

Chapter 1

Introduction

1.1 Motivation

The web is a free community where no ruler exists. Anyone, with the knowledge of course, can make a web page public, and no one can arrest them for not following the open standards of HTML coding. Due to this, many web pages contains errors compared to the standard. Luckily, it is in everybody's interest to let the page follow the defined standards. But some times the author does not have the knowledge to fix the errors, and other times he is satisfied because the page is viewable in most of the cases. They forget the small minorities like those who use other operating systems (OS), other web browsers or other viewing devices. An example of the last group is blind people, who use a board (Braille device) which they can read with instead of a screen.

This Braille device can obviously not show images, and is therefore limited. Mathematical formulas, and text in a fancy font, are often represented in graphical formats. That is a non adequate solution, because it limits the use for blind people or users with a text based web browser (typically lynx).

The normal way to write mathematical formulas in academic environments is LaTeX or just TeX. From a LaTeX document a Postscript or PDF file can easily be made. Several methods of how to publish mathematical formulas on web are listed in [3]. Two of them are LaTeX to HTML translators. The first of them translates the formulas without removing the availability, but the result is a limited so e.g an vector (letter with arrow on top) cannot be represented. The other translates the formulas into graphics, and thus makes it less available.

Pure HTML can in a large extent be used to write formulas by using the special signs (e.g. `p` equals π), but for more

advanced equation a better tool is needed. MathML is an open standard defined by the World Wide Web Consortium (W3C), and version 2.0 got a W3C Recommendation on the beginning of 2003. Tim Berners-Lee, the Director in W3C is quoted very elegantly in [4] where he comments MathML:

MathML will make the Web even better for educational, scientific and technical materials. It also has the potential to make mathematics accessible to those with visual disabilities. It will allow mathematical content to be reused and exchanged with technical computing systems for further manipulation.

For now Mozilla, and the web browsers in its family, are the only ones that has support for MathML. An example of MathML in Mozilla is shown in figure 1.1.

MathML is definitely the preferred way to represent formulas on the web, because advanced formulas can be written, and they can be read by everybody, even text-based web browsers (but of course they won't have the fancy style as in Mozilla). For more information about MathML it is referred to the homepage to MathML in [5].

$$\begin{array}{c} 2 + 3 \\ f(x) \\ \sin(x) \\ a(b+c) \\ ab+ac = a(b+c) \\ \int_0^b f(x)dx \\ \frac{d}{dx} f(x) \end{array}$$

Figure 1.1: MathML shown in Mozilla

1.2 The objectives of this project

This project is named "Web Content Mining", and is given as an assignment in the course "Web Mining". The assignment is to create a web crawler that downloads the images in a web page and tries to categorise the content of each image into different categories, e.g. mathematical formula, logo, buttons, etc. The focus is on automatic detection of image usage that reduces the accessibility of a web page. The last sentence is directed towards text and mathematical formulas which are represented as images.

From this assignment description we interpret that the main focus is to differ between text (including formulas) and the rest, which are image usage

that reduces the accessibility. In addition we have focused on separating *real photos* and *graphics* like logo and buttons. Realistic photos (like 3D-modelling that use advanced algorithms like ray-tracing) and game screen shots (e.g. Doom 3) are very similar to the real photos category. It is not emphasised to differ this category from *real photos*, because it is of no real use to know the difference. It is also not focused to differ between the different graphic objects e.g. logo and buttons, because those categories are quite similar and it is no use to know that difference either.

The crawler is created with focus on execution speed. Algorithms that separate two close-lying categories will in most cases be very slow. This is the other reason for not to differ between photo and realistic image, and logo and button.

1.3 The structure of this report

The approaches and algorithms that may be used to solve the task are listed and discussed in next chapter. Chapter three defines a test-bed that tests the algorithms presented in chapter two. The algorithms are tested on images from the categories, and the result is presented in graphs so the difference of the categories easily can be seen. The categories tested are photo, realistic, graphic and text.

Chapter four combines the approaches and algorithms from the second chapter with the test results from chapter three, to make the best solution to solve the given assignment. The final solution is tested at random web pages, and then the images are manually control checked afterwords. Those data are used to check how exact the crawler is, to fix bugs, and to fine adjust the settings. The optimising process is based on the *try and fail* method.

Chapter five makes up the summary, which is some finishing words about the results, the testing process, and future work.

Chapter 2

Categorising approaches and algorithms

In this chapter theories and algorithms are presented, that are thought to be good to categorise graphic. The first section is about the approaches that are used in the algorithms or directly in the process of categorisation. The difference between an approach and an algorithm is subtle, but approach is used on methods or tools (or very small algorithms) that the programming language offers. The algorithm is the sequence we write. This is clarified further out in this chapter, but a small example of an approach is using the size of an image to decide if the image is big enough to be counted as a real photo. An example of an algorithm is a method that calculate an image's *entropy*. Only the algorithms with the biggest probability to succeed are presented here.

2.1 Categorising approaches

2.1.1 Format

As far as it concerns us, there are 3 different image formats. Raster, bitmap and *lossy* formats. Since all pictures are converted to bitmaps before we start analysing, there is no real problem with the two first. *Lossy* formats such as jpeg are the thing that can provide trouble. Because of the compression algorithm used, some information in the image will be lost and therefore result in graphical artifacts of the image. This may in some cases cause faulty results in the categorising tasks. This is mostly because jpeg uses an approach that makes strong contrasts in the images blurry. The only way to correct this is using filters and try to sharpen up the images, but this is

an approach we choose not to spend any effort develop since in reality the information is already lost. But the widely used formats PNG and GIF is using indexes. This is excellent for our purpose and the only reasonable format to use on text and formula.

2.1.2 Size

The size of the image is an property that is easy available in most programming language, and it can fast exclude some categories. If an image is more narrow than 8 pixels, it is presumably not a text (we consider 8 pt text as a minimum) or real photo. The image is probably a *design element*, e.g. a border, separator or similar. Icons, which we have defined as images smaller then 20×20 , are also possible to exclude.

Really big images can also be excluded to lower the processing time. If the categorising were to be used in real-time, big picture (typically over 1280×1024) would not be processed quick enough and should therefore be excluded.

2.1.3 Histogram

An image's histogram is well known for people who works much with image processing programs. The histogram describes how many pixels per colour an image has. It is often represented as a graph in an *Image Editing* program. The X-axis is then the colour value, and the Y-axis is the frequency. Bright colours usually have a low colour value, and dark colours have a high colour value. So if the graph has one peak close to the Y-axis, the image contains many bright colours.

The histogram is an essential tool in the categorising process. The histogram is used to quickly get values like sum of all pixel values (called sum of colour values), most used colour, number of colour used, etc.. The histogram is used in almost all of the algorithms, either directly or indirectly through the values that comes from the histogram.

The histogram is defined differently in the different programming languages. In Python (which we use) the histogram from a "RGB" picture is a concatenation of each band's (one band for each colour, R , G , B) histogram (RRR..GGG..BBB..). In C, the histogram consists of a weave of each band's histogram (RGBRGRGBRGRGB...). A third option is to add each band's histogram to one, such that a grey shade histogram is achieved ($R+G+B$, $R+G+B$, ...). If a histogram function is not supported by the programming language, it is fairly simple to make it, based on the pixel values. The result will of course vary based on which histogram that is used.

Histogram peaks

If the histogram has only one peak, the image is or one colour. If the image consists of two colours (which text and formulas do) the histogram would have two peaks. These peaks may be ragged and have side values which are high, caused by anti-aliasing and such.

2.1.4 Image conversion

The image can be converted between different *modes*. A mode of an image describes the way it represents colours. In Python, each mode is represented by a string, and can be for example "RGB", "CMYK" and "1". Image conversion can be used to get an image with conform settings and properties, so it is easier to process. Conform images can be processed the same way.

The anti aliasing effect can be reduced by removing less significant bit in the colour value. Similar colours are then counted as one. An image conversion into the mode "1" (monochrome) will also lead to the same result, but the algorithm is not so exact in the conversion. Typically red or yellow text on black background can loose the text information when converted into monochrome. A function like conversion is not so much emphasised to keep its information that lies in between.

2.1.5 Crop image into sub image

One big image can be analysed, and then cut or cropped into several smaller images. The cutting can just be done systematically not based at the image information at all, or it can be based on a pre analysed. The analyse might register the background colour, and cut it away, in such a way that the essential is left back. It might also separate the image into a part that have realistic properties and one part that has graphical designs.

We have only implemented a systematically cutting algorithm, which is named *Poisson*.

2.2 Algorithms

2.2.1 Standard deviation

Deviation is a statistical term which express how distributed the single values are in comparison to the mean value. The formula is $std = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$ where \bar{x} is the mean value. The standard deviation will usually be a high

number in an area between zero and the mean value. The mean value is the average colour value of each pixel.

Both values, standard deviation and mean value, can be calculated from a histogram that is presented in present instead of number of colour values. With other words, the histogram is divided on the total number of colour values. This will give a result that does not depend on the *mode* of the image. The results are normalised. This algorithm will probably split single coloured images or images with many similar colours from images which contains a broad spectrum of colours.

2.2.2 Entropy

The entropy is used frequently as a synonym for *disorder* and entropy is a term in thermodynamics, statistical mechanics, and information theory. It describes random events, and how random they are. In [7] is information theory defined as a measure of the average information rate of a message or language. The entropy is given with the formula $-\sum p_i \log(p_i)$ where p_i are the probabilities of occurrence of the symbols of which the message is composed. The entropy represents the average information rate per symbol. In a more mathematical sense and wider use entropy is any quantity having properties analogous to those of the physical quantity, especially the quantity $-\sum x_i \log(x_i)$ of a distribution $\{x_1, x_2, \dots, x_n\}$.

The entropy of an image will hopefully give high values to real images and low values to text and graphics (constructed images).

2.2.3 Pixel search

We have chosen to call this method line-scan, after the way it processes images. The theory is that images of text will have two colours which makes up most of the image. These two colours are then scanned for the blank lines which appears in text between lines and characters/words.

At first the image is converted to a 8-bit gray-scale image. This is done so that we don't have to process the image in 3 different colour-bands. Additionally the chance that some pixels of the a different colour could end up being converted to the same colour decreases. But this error would be partly corrected by the other algorithms.

Thereafter the histogram of the image is searched and the two highest colour values are stored. The values are marked as background colour and foreground colour. If these makes up less than 85% of the image this method concludes with the fact that this are not text and passes this value back. The picture are now scanned vertically for lines that contain less than a given

percentage of the background colour. This scan returns a list containing top and bottom vertical position of potential strings. If there is a distinct number of these lines a percentage that the image is text is added.

Afterwards these lines are scanned horizontally one after another. We now have a 4-point value specifying each character. If there are enough of these we add a weighted chance that we are dealing with an image of text.

2.2.4 Pattern recognition with Poisson

In [6] a method is described of how to decide if the distribution of objects distribution are random, regular, or in groups, where regular distribution is a pattern. The calculation is based on the previous described algorithm with *standard deviation* and *mean value*. The statistic value *variance* is the square of the *standard deviation*, and it is that value that is used.

The area (in this case the image) is split in numerous boxes (smaller images). In each of these boxes the sum of objects (colour values) is counted. Then the *mean value* and *variance* to the number of objects per box are calculated. If the *mean value* is more than the *variance*, then the objects are regular distributed. The opposite alternative suggests a group distribution of the objects, and when the values are close to equal (which is a Poisson distribution) the objects are random distributed.

The size of the boxes is a vital part of the result. A quick reasoning gives the following extrema cases in box size. If the boxes are as small as one pixel, the sum of the colour values in one box will then be the colour value of this one pixel. The *variance* and the *mean value* is then the same as on the whole image. If the box is defined as big as the image, it will only be one box. The *mean value* is then the sum of all the colour values in the picture, with other words a very high value. The *variance* of one box is zero (look at the formula for standard deviation in chapter 2.2.1 to get the proof).

The problem is to find the perfect box size to use, which is difficult because a photo can be taken from close distances and far away. Text can also be big or small, which leads to the conclusion that the box size should be dynamic. In figure 2.1 the result of the Poisson algorithm with variable box size is presented.

The scales are not equal, but it is easy to see a clear difference. Unfortunately a test like this is too time consuming to be a part of the final result. It is difficult to conclude with anything, but a box size on 8-10 will give the clearest difference.

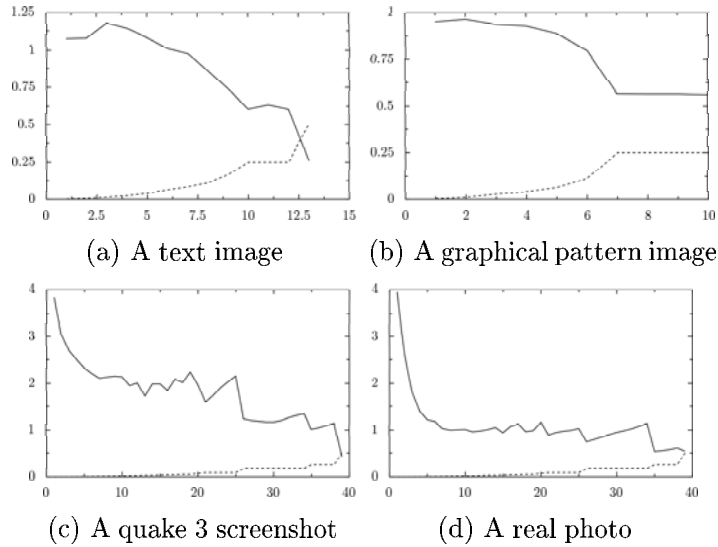


Figure 2.1: Poisson results with different box sizes. The box vary from small to big along the X-axis. One side of the square boxes is five times bigger then the X-value. The dotted line is the *mean value* and the solid line is the *variance*

2.2.5 Benford's law

Dr. Frank Benford noticed that pages of logarithms corresponding to numbers starting with the numeral 1 were much dirtier and more worn than other pages. Benford's law is based on the fact that the number *one* is very frequent, and in fact more frequent than other numbers. Based on mathematical statistics, it is natural to believe that 1 is just as probably as 7, namely $1/10$. Dr. Benford showed that 1 has a probability of around 0,33. This can be read more about in [1] or other sources to get the full logic in it, but the main point is that this law can be used in image categorising.

In [2] Alan DeKok has written a program in *C* that uses Benford's formula to separate real photos from unreal (graphics, etc.). The program is based on the theory that real photos follow the formula better then graphics and constructed images. This program is based on histogram folding, and it works surprisingly good.

Chapter 3

The testbed

The testbed use hand-picked images locally, so the download-time won't be a factor.

3.1 Test method

The efficiency of the algorithms, measured in exactness and speed, were tested in a test program. The program used hand-picked pictures located locally on the hard disk. The pictures were manually sorted in the categories earlier defined. Then the program analyses each picture with only one algorithm. The results are then returned in numbers, which are presented in a bar-graph so it is easy to measure the efficiency. The graph is represented with the picture number along the X-axis and the results in number along the Y-axis. Then there are one bar for each category. This way it is easy to detect algorithms which clearly separates the different categories. The time the algorithm use is also measured. The time measured is the time it takes to get the results plus the time it takes to generate the graphs. The graph generation time is about 2.5s for each graph which sums up to 10s for each test and there are in total 81 images in the test. All of different sizes and formats.

3.2 Test results

The tests were studied and discussed internally until we came up with an conclusion that both could agree with. We have tried to collect very different images in each category and by this we hoped to get things as realistic as possible. This makes some of the results behave as offsprings. In-fact, some

of them may better be placed in other classes after further study. These results are used as an basis for the weighting of the algorithms.

3.2.1 Entropy

This is the fastest method and was completed in 28sec. It gives about the

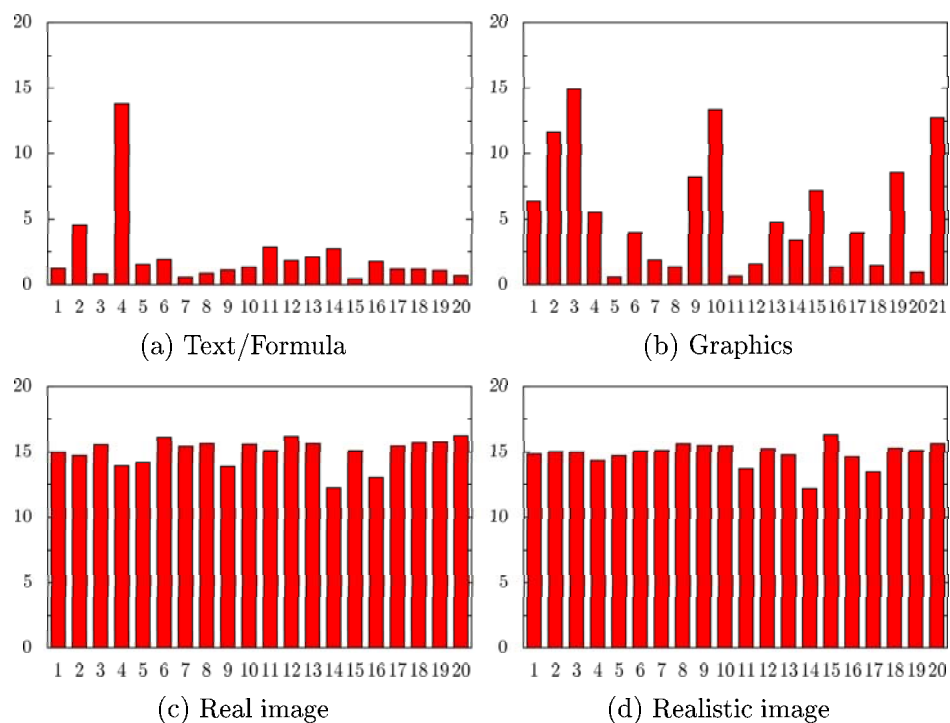


Figure 3.1: Entropy test results

same results as Benford, only in a little lower resolution. Its therefor a better method to use since it takes up less CPU time. But in cases of doubt Benford is to be used.

3.2.2 Linescan

The Linescan completed in 62sec but it is hard to compare the time with the other times since Linescan do not spend any time analysing images if there are no data in the image which implies that its a text image. This makes real and realistic images process really fast while images with many text-lines will take a long time. The results shows that it acts really good on text images,

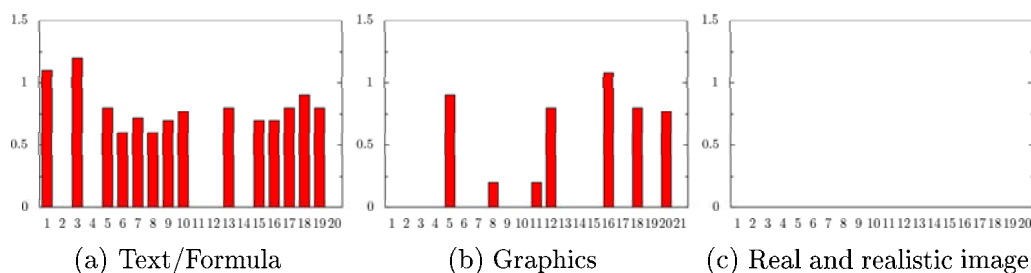


Figure 3.2: Line-scan test results. Real and realistic image have an equal graph.

but not that good on formulas. It has little use in determining whether an image is a graphic element or a real/realistic image.

3.2.3 Poisson

Poisson takes 58sec to complete which is fair enough. In this method there

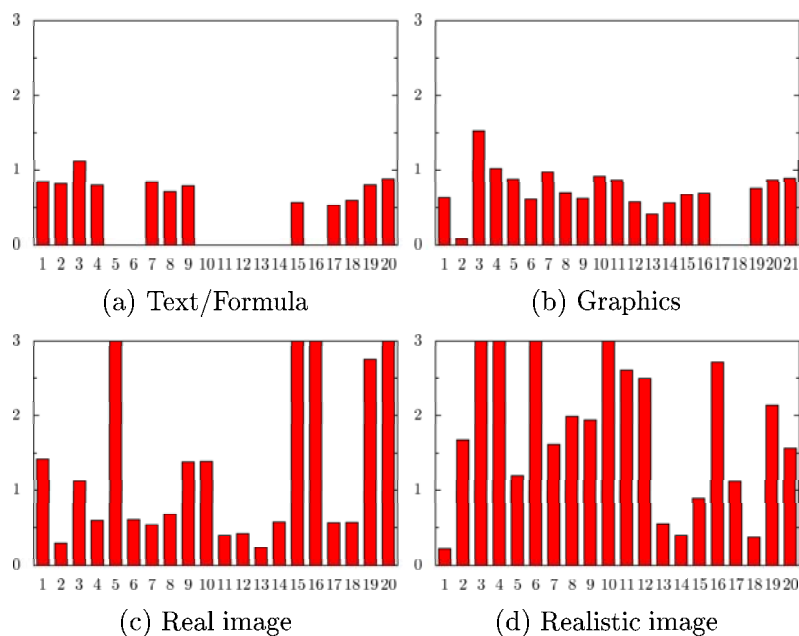


Figure 3.3: Poisson test results

are few results that really stand out, but it seems clear that only real images get really high results. The other values are too uncertain to have any real

usage in our tests.

3.2.4 Benford

This is by far the slowest of our classes in the testbed, it took 923sec to complete. And is therefore not too useful for our project. But it usually gives very good results. So it is to be used only if there are great doubts after the Entropy scan. It is easy to see that real and realistic images separate very

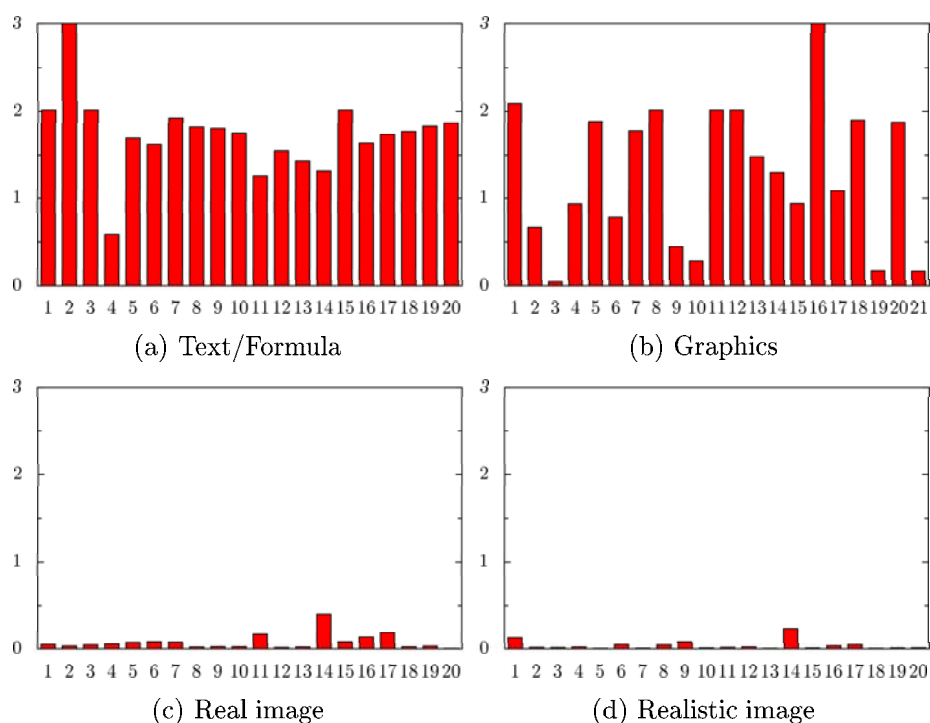


Figure 3.4: Benford test results

clearly from the rest, but there is no clear line between text and graphics. This means that this method is good for finding real and realistic images. It also means that very low values implies either simple graphics or text, but it's impossible to be certain which. If the second pass in Benford was to be used it would make it possible to distinguish real images from realistic images. But since the method is so slow and it's really out of the scope of this assignment we chose not to use it.

3.2.5 Standard deviation

Standard deviation turned out to not give any good results at all. This is caused by the algorithms nature and the categories. If an image as a big standard deviation it means the image has big contrasts, while a small standard deviation means the colours to the image are in the similar. Out from this it is impossible to decide in which category the image belongs. But the standard deviation is used indirectly in Poisson, and it works satisfactorily.

Chapter 4

The final implementation

This chapter covers the crawler and its algorithms for categorising. The algorithms are chosen, based on the test results in chapter three. The optimal combination of algorithms, that makes the processing time as low as possible without ruining the high exactness that is needed to categorise properly, is sought.

The crawler is then tested on random web pages to see how it function in practice. The images it categorise are controlled manually by humans afterwards so errors can be revealed. If serious errors arise, the solution must be evaluated and redesigned. The final solution is at the end presented with its design, algorithms and test result. But first the crawler, which collects the images on the web pages, is covered.

4.1 The crawler with its web mining part

(The image retriever) We thought the making of the crawler, which traverses the web pages, would be an easy task. But it turned out to be quite complex because of Java-script and other non-HTML syntaxes in web pages. The HTML package in python threw an error each time it met non-HTML syntax, and consequently did not parse more on the page. At first we tried to correct this by using regexp to dispose the garbage in the code, but even advanced regexp was not sufficient to get rid of the faulty elements. A pure regexp crawler was made, put the syntaxes used in some of the pages just was to complicated for this solution.

The solution was to expand the built-in SGML parser (SGML is the "father" of HTML and XML) shipped with python. This parser was expander to just ignore all tags but the ones starting with "<a" or "<img". By doing this, all the invalid HTML tags are just removed and the need for string

replace in tags is eliminated. It also convert local links to a full URL containing site name. This is needed since we have got to have a reference to the link. A link can contain either a image or a link to another site, this is later processed in another class. To the best of our knowledge the crawler now works flawless, even on pages full of weird code like e.g. `http://www.vg.no`. The mime-type is checked for every link to determine if it is an image or just another link, images are passed over to the categorising class, which we will cover in the next section.

The crawler we made works like this: An starting web page and a number which describes the depth of of the mining, is given to the crawler. The crawler traverses this page and picks out all images on this page. Each image is processed as soon as it is traversed. Then, all the links, starting with the first, are opened. The web crawler has two input values, namely a starting web page and a number which describes the depth of the traversal. The crawler starts on the starting web page and categorise each image on the page. Then all the links, starting with the first, are called up recursively. The number of depth traversal is then decreased with one. When it is equal zero, the links on the page are not traversed.

The crawler can easily be extended with more directional factors, such as only to traverse web pages within the domene of the start page. Due to the assignment given, with its time limit, this was considered out of this projects scope. Our focus were on the image categorising.

4.2 PictureContainer

This class is responsible for handling all the data and evaluating it.

When a URL is added to the container the class downloads the image and checks the md5 sum of the image up against an internal dictionary. Since processing the images often may take some time this eliminates the need for processing the same image over and over again. Re-usage of images are a typical thing to do, especially when it comes to *design elements* on web-pages.

The image is then passed on to the categoriser described in the next chapter. The categoriser then returns an integer value of 0, 1, 2 or 3. 1 means that the image is evaluated to be text, 2 means that its an *design element*, 3 is a real or realistic image and the return-value 0 means that the image is simply to big to be processed. The URL and result is then added to the dictionary with the md5sum as key.

There is also a possibility to save this list as a simple formatted web-page. This was in first place meant to give us an easy way to evaluate the result of

the categoriser. The is separated in lines where each line shows the analysis of one image. The line contains 3 elements. The result of the analysis, the image itself and a hyperlink of the URL to the image.

4.3 The categoriser

The categoriser is the worker of this project. It is responsible for calling all the analysing classes and weighting the results to make out what kind of image it is. The results are made up by percentage chances. This means that the type with the highest percentage is the one we return. Its worth to note that due to the nature of the weighting the percentage may or may not add up to 100%.

The first pass is an analysis using Sizeexclusion. This is used to find images that have sizes which automatically makes them an *design element*. This class is also used to find images that simply is to big to do any analysis on.

Entropy is the next class used to analyse. This method is very good at determining that a picture is not a real image or not a text image. But it are not good at determining what class the image belongs in. This means that it returns a weighted chance to which class a image belongs to. As an example its possible that it returns that an image has 30% chance of being an *design element* and 60% chance of being an real image.

It the result of the Entropy test is very vague we pass on to the time-consuming Benford test. This test is very much like Entropy test but gives a bit more accurate result. Because of this, the test is weighted very much in the same way as Entropy. It is worth to mention that this test is not used instead of Entropy, but in addition to it.

Linescan is used as the next test. This is a pure graphical method of analysing and because of this ability its very good at recognising text, but not any good at determining if an image are an *design element* or a real image. As an result of this the test is weighted very high if the result is text, but *design elements* or a real images are weighted almost equally low.

As an last test we use Poisson to give real images a little more chance to appear. This test don't work on all real images and hardly at all on the other types of images, and is therefor weighted very low.

After this the class returns the most probable image-type. The tuning of the weighting is the thing that will take the most time to get real good. As it is we have tuned it using graph output and trial and error. This is an time-consuming process and our result is sure to be even better if we spent a lot more time tuning on this.

Chapter 5

Summary

Wrong use of images on the web cause decreased availability and discrimination of some minorities. The most important objective with this project was to identify use of images to represent text and mathematical formulas. That we have achieved.

The diversity of images is huge. As seen, there are some characteristic too each category of images, but the borders that separates the categories are very vague. An example of that is a logo which consists of only text with fancy font and colours. Should that be categorised as text or graphic, and how many colours can a text have before it is a logo? Another good example is an image which is a photo of text. If this photo is taken really well, in such a way that the result is like a scanned document, it could be impossible, even for the human eye, to tell if it is a photo or text. As said, the diversity of images makes it impossible to categorise 100 % correct.

Our final prototype has a very good hit-rate, and it is not *fine tuned* yet. That means it is room for improvements, and even better categorising.

The *find tuning* consists of testing the crawler and to adjust the weighting of the algorithms. The user can also weight the algorithms after own preferences in so the "correct" category is chosen in doubt-cases. A doubt case is for example a hybrid image with graphical elements in a photo. Other future work areas is of course to find new algorithms and approaches, but that could be hard to find. Some optimisations of the ones that are implemented now is more likely to succeed. The speed of the crawler can also be optimised, so it can better be used in real time.

Blind people can not read the images on web pages, and too often is the naming of the images not descriptive at all. The crawler could possible be rebuilt to work as a plug-in in a web browser, and report what kind of category an image is, in addition to its name. But then it is important that the speed is good enough. Another use, is in the image search to yahoo.

Currently it is possible to choose between *photos* and *graphics*, but it works like a joke. A light weight version of the code in our crawler would be quick enough for this kind of search, and work far better.

The main use of this crawler is of course to detect wrong usage of images on the web. This function is intact and work very good.

Bibliography

- [1] Benford article, www.rexswain.com/benford.
- [2] Isreal - a picture analysis tool, <http://www.striker.ottawa.on.ca/aland/isreal/>.
- [3] Mathematical formulae, <http://www.univie.ac.at/future.media/moe/formeln.html>.
- [4] Mathml by w3c - faq, <http://www.w3.org/math/mathml/faq.html>.
- [5] Mathml by w3c, <http://www.w3.org/math/>.
- [6] P. C. Hagen. *Innføring i sannsynlighetsregning og statistikk*, volume 2. Cappelen akademisk forlag, Oslo, Norway, 1998.
- [7] E. S. C. W. J. A. Simposon. *Oxford English dictionary*, volume 2. Oxford, England, 1989.