# Information extraction and machine learning

**Tomas Malmer**
tf06tm0@student.lth.se

## Abstract

During this project a multithreaded web crawler and information gathering framework named CrawlingMowgli has been developed. The main focus has been on getting a framework that is easy to extend and handle a lot off different types of entities that are found on the web. Some techniques to improve the result and extraction quality by using machine learning algorithms were also explored and evaluated.

## 1 Introduction

The purpose of this project has been to understand how different approaches in information extraction can be used in order to extract entities from web sources and implement a prototype. The resulting prototype is called CrawlingMowgli and consists of two parts, the Crawling engine and the extraction engine Mowgli. The core part of the extraction engine is built upon regular expression but some other functions were implemented like trigger patterns and logic to take advantage of the structure of a HTML document. Information extraction from real world data can however be be a great challenge since the raw data contains a lot of strange things, everything from malformed HTML to very special presentation of the information.

In order to evaluate the performance of the engine the result from extracting phone numbers from company websites was compared with the data from blocket.se.

## 2 Architecture

The crawler fetches a webpage from internet and parses the HTML. The parsed page is then used to extract information as entities that are organized in a database.

$$\begin{tikzcd} Crawler \arrow[r] & Extraction \arrow[r] & Organization \end{tikzcd}$$

### 2.1 Web crawler

The web crawler was developed especially for the task of this project by combining libraries in python for fetching webpages and the open source libraries for parsing HTML (Beautiful Soap). The first version of the crawler was single threaded and took more then an hour to crawl all the 923 companies in the testset. When the structure of the program was changed to support multiple threads there was a big performance gain.

The features of the crawler are pretty basic, each thread pops a site address that should be crawled from a queue and follows every "interesting" link found in the page. If a link is interesting or not is determined by check if the link matches certain regular expression, for example *'kontakt|om\s|contact|about|om[_\s-]*oss"*.

## 3 Extraction algorithm

In order to extract information from a web page and organize it in meaningful manner, one has to go though certain steps. You will need to ...

- Find the document which contains the information.

- Locate where the information is in the document.
- Extract it.
- Normalize it.
- Store it in a appropriate manner.

HTML has some extra properties compared to normal corpus that can be used to improve the number of “good” extractions. One such property is that the information inside the document is structured inside tags in a hierarchical way. That can be used as an indicator of how different blocks of text are related. The main ideas are to ...

- Take advantage of the structure of a HTML page.
- Use combinations of regular expression to extract the interesting parts.
- Design a system that can be extended and trained by using machine learning techniques.

### 3.1 Inspiration

The algorithm was inspired by how humans search though text to quickly find an answer. Usually we look for keywords and things that “looks” in a specific way, for examples a lot of numbers grouped together is easy to spot in a text with a lot of characters. This is because it stands out from the rest and it is a pattern that is easy to separate from the rest of the content.

The same information could be obtained by slowly read through the text and the result would probably be better but you will need more concentration or time. To emulate this behavior in a computer is quite complex, therefor the first a approach was chosen since it is easier. An illiterate or a child could probably mark the correct things in the text using this technique since you don’t have to understand what you read. The decision is made by just looking how things are grouped together in a small context. It is this property thats makes it a good starting point to begin to implement it on a computer and take it further from there, in iterations. Just as children learns more about the world with every experience.

### 3.2 Triggers

Inspired by the idea of human reading behavior mentioned above, tigger patterns was born. When I read scan though a web page looking for a telephone number I first look for a link with a text close to “Contact us” or something similar. I then click the link and search for part of the page with a lot of numbers formatted in a phone-like way. If the text “Phone:” is close to the number sequence, it’s almost for sure that the sequence are a real phone number. Triggers removes some of the false matches that would have been found by just using a regular expression looking for digit based phone numbers.

The trigger patterns where implemented by using various regular expression that where constructed by hand in since I didn’t have time to implement something more advanced. Constructing the trigger patterns by hand was a good experience seen I gained a deeper understanding for the dataset which could come to use later on.

An example of a trigging expression for a swedish phone numbers is *telefon|tel[:.\s]|(\+46)|växel*.

### 3.3 Extraction

When a trigger is executed, all the extracting regular expression are fired against the current element of the trigger. If the extraction pattern couldn’t be found inside the current element, the search is repeated for the parent element until something is found or the maximum hieght is reached. Triggers and extraction patterns are perfect if the information is structured in a table with a label in the left column that hopefully will be matched by the trigger regular expression. The content in the right column will then be extracted by the extraction pattern that are associated with the trigger because of the graph traversing.

An example of a extraction pattern that where used to extract phone numbers from swedish sites is *[0-9+]{3,5}[()\s0-9-]{3,}[0-9]{2,}*. The pattern was fires after the phone number trigger as been fired.

## 4 Implementation

Everything was implemented in Python because of all the libraries and tools that are needed were available for free and its a good language to get some more experience in. The parsing of the HTML-page was done with BeautifulSoap which builds a graph that can be traversed. It is also rather good at parsing malformed HTML-pages which is important in a real world application, especially version 3.0.8. Python also the a full featured regular expression package that contains everything expected of a modern regular expression engine.

## 4.1 Mowgli - The extraction and learning engine

The extraction and learning package that was done during this project was called Mowgli. An engine that hopefully will grow up in the information jungle and one day become "man".

## 4.2 Learning regular expression

Something that would increase the performance and maintainability of Mowgli is some way to make it learn regular expressions by positive and negative examples. There has been work on this subject by others, especially (Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, 2008) which inspired the first approach to the problem.

The algorithm makes use of a number of transformation rules that makes transformations on a given regular expression. A first guess of the searched regular expression is given as starting input to the algorithm. In each iteration of the algorithm, all transformations rules as tested and the best one is chosen according to the F-score in a greedy manner. The loop continues until there is no improvement on the testset. A simple transformation could be to change the number of times a pattern should be matched. For example, the pattern *[a-z]* could be transformed to *[a-z]{1,1}*, *[a-z]{1,2}*, *[a-z]{1,3}*, *[a-z]{2,3}* and so forth, looping though all combinations between a minimum and maximum length. Another kind of transformation is the character class expansion, for example: *\s* could be expanded to all whitespace characters in the set *[ \t \n \r \f \v]*.

My first attempt on implementing the algorithm was to apply the transformation rules by searching and replacing on the current regular expression string. Under the development I realized that the structure I choose to implement the transformation rules had its limits. Simple string search and replace works for "simple" input regular expressions but when the input gets more advanced you would have to use regular expressions to make the transformation which would reduce the performance and be hard to maintain in the long run. A better approach would be to build an own representation graph of parts that builds up the regular expression. By parsing the original regular expression in the beginning of the algorithm would allow the elements to be "normalized" and constructed in such a way that the transformation are easy to make. It would then be possible to do things inspired by evolutionary algorithms, collect statistics on the transformation rules and combination in a neat way. This may, or may not, improve the speed of the algorithm.

Unfortunately did I not have time to complete the alternative implementation during this project but it will eventually be implemented afterwards.

## 4.3 CrawlingMowgli

The web crawler combined with the extraction engine Mowgli is called CrawlingMowgli and is ready to handle real webpages. The test of the performance was done against blocket.se, a swedish site that contains a lot small companies that put their adverts on the site. The websites for these companies are listed on blocket.se together with their phone number and some additional information. By extracting the correct phone numbers from blocket.se and crawl the companies homepages with CrawlingMowgli it is possible to get a rough estimation of the extraction performance by comparing the extracted phone numbers against the one listed on blocket.se.

The sites crawled in the testset was not alway very "professional" and contains a lot of malformed HTML and strange design. Because of this selection of webpages the following result should be seen as a lower bound for the performance of CrawlingMowgli.

## 4.4 Results

This is the result of the testset from blocket.se. The entities that were extracted where phone number and email.

|  |  |
|---|---|
| Total number of companies | 928 |
| Successfully parsed the homepages | 89.3 % |
| Recall | 30.1 % |
| Extracted some entities | 52.0 % |
| Extracted phone entity | 45.7 % |

Blocket.se did only list the main phone number to the company and since many homepages have phone numbers to different parts of their organization a lot of extra phone numbers are found. This makes it difficult to estimate the precision of the algorithm with the current testset.

### 4.5 Improvements

There are a lot of room for improvements. One of the more interesting areas that could improve the result significant is the machine learning parts of CrawlingMowgli that currently are very basic. By collecting statistics on the elements near the entities that are found in a HTML pages it would be possible to construct better triggers. Maybe by using something inspired by tf-idf or more advanced methods.

Not all of the proposed transformation rules in the training algorithm for regular expressions proposed by (Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, 2008) where implemented. One of these rules was the negative lookahead transformation that can negate a positive match if a certain pattern are proceeding the original pattern. If it's better to add this transformation to the extraction pattern or to implement negative triggers is something that needs some more research but it will almost certain improve the end results' precision.

Another function that could be implemented around the triggers are different weights for different HTML element containers. For example, if a trigger is found inside a table it would cost more to try to extract something outside that table. This is since the probability for something that is inside a table would have connections to something outside of it is low. This way, different HTML elements could be seen as "information energy potentials" that you need a certain energy in order to escape from it. Extraction can only be made up to that energy level that is associated with the trigger that was fired.

Many entities contains some parts that can be matched agains a database as a part of the validation of an entity candidate. For example, many phone numbers often contains an area code that can be matched agains a database. By checking if a phone-like number sequence contains an area code would be a strong indicator that this number sequence really is a phone number.

## References

Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan. 2008. *Regular Expression Learning for Information Extraction.* IBM Almaden Research Center. San Jose, CA 95120 `http://www.aclweb.org/anthology-new/D/D08/D08-1003.pdf`.

Beautiful Soap. *HTML parsing library.* `http://www.crummy.com/software/BeautifulSoup/`.

Pylons *A framework for building websites.* `http://pylonshq.com/`.

Blocket.se. *A swedish site with adverts and companies.* `http://www.blocket.se`.

Python 2.6. *A programming language.* `http://www.python.org/`.