AUTOMATED WEB PAGE CATEGORIZATION TOOL

Radek Lát
Master Degree Programme (3), FIT BUT
E-mail: xlatra00@stud.fit.vutbr.cz

Supervised by: Dominik Malčík
E-mail: imalcik@fit.vutbr.cz

Abstract: This paper describes design and implementation of a tool for automated web pages categorization. The goal of this tool is to be able to learn from given sample web pages how each category looks like and later to assign these categories to previously unseen web pages, support any reasonable number of categories, and support multiple languages. This tool uses advanced machine learning, language detection, and data mining techniques, it is based on open source libraries and it is written in Python 3.3.

Keywords: machine learning, SVM, crawler, language detection, categorization

1 INTRODUCTION

With the current widespread use of the Internet, it becomes more important for companies to control what online content can their employees to access. One way of limiting access to unwanted content is by blocking certain domains, possibly categorized into pre-defined categories. However, due to the size and rapidly ever-changing nature of today's Internet, it is humanly impossible to categorize web pages by hand.

To automate the categorization task, it is possible to use advanced machine learning principles that have undergone an extensive research in the past few decades. Majority of this research uses textual information from the web pages. For this task, the current state-of-art algorithm is the Support Vector Machine (SVM) [1, 164], [2, 408]. Typical research using SVMs for web categorization uses sample data collections, such as Reuters-21578, WebKB or similar. However, these sets do not reflect the current state of the Internet. Real web pages have often broken HTML code, contain many irrelevant and noisy data, hide their content into `<iframe>` HTML tags or external JavaScript source codes, or contain multiple languages.

The tool designed and described in this document addresses these problems. In other research papers dealing with web page categorization, authors use large-scale web crawlers and explore the web as a graph, using the discovered structures as features for categorization. On the other hand, the tool described in this document tries to use purely the front page of a web to allow quick categorization at the cost of possibly lower precision. The primary goal is to be able to categorize web pages into "pornography", "gambling", "shopping", and "none" categories in English, German, and Czech language. Training data are manually collected and verified.

2 PROBLEMS WITH THE DATA

2.1 FETCHING WEB PAGES

First problem to address when categorizing web pages is how to download web pages. Some of the common problems are:

- significant network latency in web page fetching and DNS resolution
- connection errors
- lost, missing, temporarily unavailable, or invalid web pages
- redirections and redirection loops

To solve the network latency issue, a web crawler based on many concurrently opened TCP connections should be used [1, 24].

One approach is using many concurrently opened TCP sockets. The advantage is speed, the disadvantage is no handling of any errors. Another approach using a web scrapping library/tool, such as *GNU Wget* (used by our tool as well), *scrapy* or similar in many threads. Advantage of this approach is already implemented handling of common network errors. Disadvantage is limited speed. Using Wget on a six-core processor in 90 threads allows us to fetch one web page each approximately 100 ms, which is acceptable.

2.2 Encoding and language detection

When a web page is fetched, it is necessary to decode it. The encoding of a web page is not known beforehand. The information present in the document's HTML code or HTTP headers about used encoding is often outdated or wrong. Therefore, encoding detection is necessary. The most accurate library for this task, that is used in many commercial software products as well, is *chardet*. Unfortunately, this library takes approximately 500 ms to detect an encoding each time. That is 10 times more than trying all possible encodings. Therefore, our tool consults *chardet* library as a last resort option.

Language detection is important for text preprocessing. One common preprocessing task is stemming, which converts words of a particular language into their root form. Another preprocessing task is stopword removal. Stopwords are words, such as prepositions or very common verbs, which have very low information value. These tasks are language dependant. There are many libraries for language detection. Among them, the *Compact Language Detector* (CLD2) library is the best. It detects over 52 languages and it is the only one allowing detection of multiple languages within one document, marking their positions.

This functionality is in our tool further extended. It is assumed that within one web page, different languages will occur in blocks. Blocks marked as an unknown language are assigned a language from their surrounding blocks, based on proportional occurrence of languages in that document.

2.3 Iframes and JavaScript

Some web pages try to hide their content into `<iframe>` HTML elements or into JavaScripts as `document.write()` statements. Sometimes, these are also combined and nested into more than two levels. Therefore, it is necessary to fetch some additional documents after a web page is parsed and these attempts of hiding a content are detected. Handling `<iframe>` tags is simple because they are just regular, already parsed HTML. On the other hand, for JavaScript there is no publicly available parser nor any virtual environment for running the JavaScript code.

Our tool takes an assumption that web pages are not very sophisticated in hiding their content and store the data as continuous strings. It uses custom JavaScript string parser that attempts to parse all strings from a JavaScript source code, while ignoring all comments. The strings are then merged and treated as HTML code. This parser is implemented as a simple general pushdown automaton.

3 CATEGORIZATION

As previously said, currently the best approach to web page categorization is using SVMs. For learning, as well as for categorization, SVM takes a set of features, which can count thousands. For text, it is common to treat each word as a separate feature. Out tool handles also words from the `<title>`, `<body>`, META keywords, META description and URL n-grams as separate features and encodes them into <*language code*><*place*><*word*> tokens. For example, a word "LearniNiNg" from the `<title>` tag becomes after processing "enTlearn", while from the `<body>` tag "enBlearn". This allows us to treat the same looking words from different languages as separate features as well.

Even though SVMs are large margin classifiers that find the right category even in very noisy data, it is still useful to perform feature selection. One of the best feature selection methods is thresholding on the value of GiniText [3]. This threshold is variable and selection of different values can be used to fine-tune performance of the classifier.

There are many approaches to weighting of words or features in general. The simplest is to treat documents as sets of words, where each word is counted as either present or not present. This is not a good representation because the valuable information about word counts is discarded. A better approach is to employ bag of words technique. One of the best according to [4] is TFRF scheme.

The tool described in this document used GiniText and TFRF as well. To compute their values, it uses the *numpy* library and to store the values a HDF5 database. All the values are stored in one matrix and computed all at once. This has been attempted also using an *SQLite* database but the time performance was not satisfactory. Computation for typical training corpus having hundreds of thousands of features would take several minutes while using *numpy* arrays takes a few seconds.

4 CONCLUSION

The designed tool is still undergoing extensive testing on different domain sets. Due to many possible variables in the machine learning process, it is still not clear if using JavaScript and `<iframe>` tags content improves precision in the categorization. However, the tool itself has already much better performance than a categorization done by hand. Some other merits of this entire project are several spin-off open source projects and fixes for the *CLD2* library.

ACKNOWLEDGEMENT

This work was partially supported by the following grant: MŠMT ED1.1.00/02.0070.

REFERENCES

[1] CHAKRABARTI, Soumen. Mining the web: discovering knowledge from hypertext data. Boston: Morgan Kaufmann, c2003. ISBN 1-55860-754-4.

[2] HAN, Jiawei, Micheline KAMBER a Jian PEI. Data mining concepts and techniques. Third edition. San Francisco: Morgan Kaufmann, 2012, 549 s. ISBN 15-586-0489-8.

[3] SHANG, Wenqian, Houkuan HUANG, Haibin ZHU, Yongmin LIN, Youli QU a Zhihai WANG. A novel feature selection algorithm for text categorization. Expert Systems with Applications. 2007, vol. 33, issue 1, s. 1-5. DOI: http://dx.doi.org/10.1016/j.eswa.2006.04.001.

[4] LAN, Man, Chew-Lim TAN, Hwee-Boon LOW a Sam-Yuan SUNG. A comprehensive comparative study on term weighting schemes for text categorization with support vector machines. Special interest tracks and posters of the 14th international conference on World Wide Web - WWW '05. 2005, s. 24-27. DOI: http://dx.doi.org/10.1145/1062745.1062854.