Automated Dataset Construction from Web Resources with Tool Kayur

Alexander Kohan*, Mitsuharu Yamamoto*, Cyrille Artho†‡
* *Department of Mathematics and Informatics, Chiba University, Chiba, Japan*
E-mail: kohan@math.s.chiba-u.ac.jp, mituharu@math.s.chiba-u.ac.jp
†*Institute of Information Technology Research*
National Institute of Advanced Industrial Science and Technology (AIST), Osaka, Japan
‡*School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden*
E-mail: artho@kth.se

***Abstract*—Many text mining tools cannot be applied directly to documents available on web pages. There are tools for fetching and preprocessing of textual data, but combining them in one working tool chain can be time consuming. The preprocessing task is even more labor-intensive if documents are located on multiple remote sources with different storage formats.**

In this paper we propose the simplification of data preparation process for cases when data come from wide range of web resources. We developed an open-sourced tool, called Kayur, that greatly minimizes time and effort required for routine data preprocessing steps, allowing to quickly proceed to the main task of data analysis. The datasets generated by the tool are ready to be loaded into a data mining workbench, such as WEKA or Carrot2, to perform classification, feature prediction, and other data mining tasks.

***Keywords*-automation; information extraction; natural language processing; web content mining**

I. Introduction

Textual information located on the Internet is usually designated for a human reader, but as the growth rate of available data is increasing, more and more automated tools are used to process such data to get insight about its properties or discover trends and patterns. Although some web resources are designed to facilitate machine processing by providing an Application Program Interface (API) to export data in structured formats, such as XML, CSV, or JSON, on many web resources useful information is still available only in HTML format. This makes automated processing more difficult, because, unlike XML, HTML tags do not describe the data they contain.

Web mining tools that extract data from HTML documents usually require a user to set up the extraction rules for each data field; the navigation rules to define transition between documents; and the integration rules to translate extracted data into the desired format. This configuration step may be labor-intensive, especially when dealing with multiple web resources of different structure. Partially because of that reason, there are web resources to which text mining has never been applied to.

Although web resources are different, some common patterns can be found in structure of information they provide. If we leverage these similarities, we may arrive at substantial simplification of web mining application to a wide range of remote sources. In this work, we argue that by imposing certain restrictions on web resources and data, web mining process can be simplified, so that dataset construction step can be accomplished quickly and conveniently.

First, we suggest that data to be analyzed are a collection of texts written in a human language, so that it can be stored in the uniform format (see Section III-A). This allows the fetched data to be stored uniformly, which improves reusability and makes possible to generate a dataset using subsets of documents that span over multiple web resources.

Second, we focus only on web resources that provide a search engine to find documents, and assign to each document a unique identifier. These conditions are met by many resources, because a web resource that provides to a user a collection of documents usually provides means to locate documents of interest as well. If these conditions are met, the navigation between documents can be automated, so that extraction and navigation rules can be simplified and their number can be substantially reduced.

We developed an open-sourced tool, called Kayur [1], to demonstrate these concepts. The tool can be applied to web resources that satisfy the mentioned conditions. It fetches documents from the Internet, translates them into the uniform format, and stores the results in a relational database. Textual data of stored documents are converted using underlying OpenNLP library [2] into a filtered list of normalized words that are then represented numerically according to the vector space model [3] with specified weight method. The resulting vectors are converted into an input format of a data mining tool (WEKA [4], Carrot2 [5], or LDA/HDP topic modeling tools by J. Chang and C. Wang [6]).

The tool demonstrates a shortcut to quickly generate a dataset for text mining needs from web resources that satisfy certain properties. Kayur requires minimum user input, has the most commonly used settings preconfigured, and provides the intuitive user interface, which makes it useful even for people unfamiliar with data preparation and preprocessing steps. By minimizing routine work, Kayur

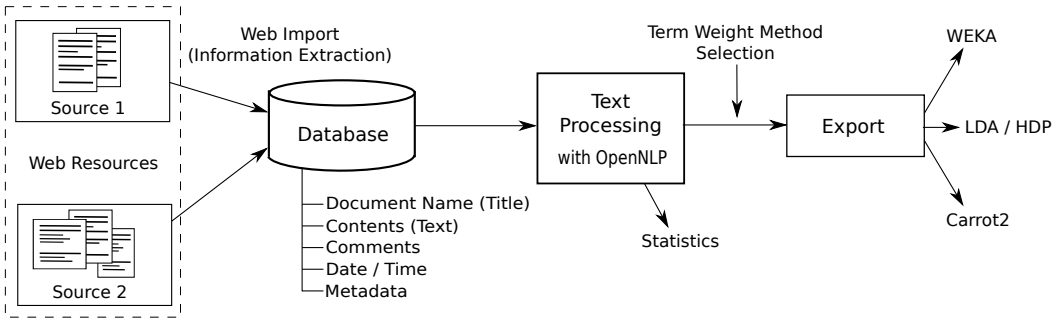

Figure 1. Workflow of Kayur

allows to quickly proceed to the most important step of data analysis. The tool itself and its source code is available on its home page [1].

The rest of this paper is structured as follows: Section II provides a brief theoretical background on text mining concepts; Section III explains Kayur's architecture and implementation; Section IV presents examples of the usage of our tool and its evaluation; Section V discusses the related projects; Section VI summarizes the obtained results and presents directions for future work.

II. Background

A. Web mining and information extraction

Data mining is the process of automatic analysis of large stores of data to discover patterns and trends. Classification, clustering, association learning, and numeric prediction are the main types of information analysis in data mining [7]. The use of data mining techniques to extract and analyze information in web documents is known as *web mining*.

Because document structure greatly differs between web resources, one of the straightforward adopted approaches to extract information was to write a stand-alone program (script) for each resource using programming language such as Perl [8]. However, better methods were developed in *web information extraction* area, and such programs were superseded with *extractors* (wrappers) that allow working with multiple web resources by describing unique properties of sources with *extraction rules* [9].

However, not all the extracted data are immediately ready to be used by a data mining tool. Numeric values and various identifiers (names, locations) can be used as is most of the time, while texts written in human languages usually require additional processing using text mining methods.

B. Text mining and the vector space model

When data mining is applied to human-written texts, the term *text mining* is usually used instead. Text mining has many important applications, including analysis of stock reports, product manuals, business and normative documents [10]. It is also proved to be useful for bug report analysis, including bug report classification [11] [12], detection of duplicates [13], and prediction of certain properties of software flaws [14].

The standard approach for applying data mining techniques to textual data is to transfer the textual data into a *vector space model*. This model represents the text as algebraic vectors in a multidimensional space, to allow calculation of similarities between documents using linear algebra operations. The dimension on the vector space equals the number of distinct terms (words) in a document collection. Each document is represented as a vector with components reflecting the frequency of a particular term in the document.

The values assigned to vector components depend on the *term weighting method* being used. For the *Boolean* method, the value of a document vector component is zero if the corresponding term does not occur in the document, and one otherwise. For the *raw frequency* method, the value assigned to a component is the number of occurrences of the corresponding term in the document.

The *term frequency* method differs from the raw frequency method in that the assigned frequencies are normalized by dividing on the maximum number of occurrences of a term in the document. The most widely used *term frequency / inverted document frequency* (TF-IDF) method considers both term frequency in the current document and its frequency in the whole document collection. There are several ways to calculate TF-IDF; Kayur uses the following formula [15]:

$$w_{ij} = tf_{ij} \times \log \frac{N}{df_i},$$

where tf_{ij} is the term frequency of the term t_i in the document d_j, df_i is the document frequency of the term t_i, and N is the number of documents in the whole collection.

III. Tool architecture and implementation

Kayur is an open-sourced cross-platform tool written in Java. It comprises: 1) the *information extraction* (web import) component to fetch data from the Internet, translate them into the uniform document format, and store the result in a database; 2) the *text processing* component to load documents from the database and process their contents;

and 3) the *export* component to convert the results of text processing into formats of data mining workbenches. The complete workflow is shown on Figure 1.

In this section, we first describe the uniform document format, and then proceed to the description of each mentioned component.

A. The uniform document format

The structure of textual data to extract differs between web resources. For example, bug reports contain attributes such as "status" or "priority", while a firm catalog has fields "firm name", "location", and "phone". Because of that, data from different sources are not usually stored uniformly.

To simplify the configuration, storage, and access to documents obtained from the web, we propose the uniform format that describes a document as having the title, content, comments, date, and metadata. These fields are chosen to be appropriate for most textual documents on the Internet. The first three fields are designated to store textual data that need to be preprocessed before they can be used in a data mining tool. The metadata field is a storage for all other parameters as name/value pairs; it is designated for data that do not require processing and can be used as is.

The main benefit of such format is that configuration process is greatly simplified, so that it can be performed even by non-trained people. Another benefit is that documents from multiple web resources are stored uniformly, and hence they can be processed and analyzed together.

B. Information extraction

The information extraction from a web resource is the first step towards dataset construction. This step often requires extensive configuration, as tools need to know the location and type of data to extract, how to navigate between documents, and what the structure of results is.

We propose a simplification of this process in the case when a web resource has two properties: a) every document in the collection has a URL that includes a unique identifier, and b) the web resource has a search engine to locate documents. Many web resources satisfy the second condition, as they are oriented for a human reader and hence provide convenient means to find documents of interest. The first property is usually satisfied for web resources that allow to view each document in a separate page.

If the mentioned conditions are satisfied, the number of extraction and navigation rules that user is required to specify can be significantly reduced. Moreover, the format of rules can also be simplified if the uniform format (discussed above) is being used.

1) Navigation rules: Suppose that the first condition is satisfied, so that the URL of each document has the form `http://...id...`, where `id` is a numeric or string identifier.

Table I
THE MINIMUM NUMBER OF PARAMETERS TO DEFINE NAVIGATION

Navigation Type	Extraction Rules	URL patterns
Incremental	0	1
Search-based	1	3

If an identifier is a number, then the navigation between documents can be simply defined by getting next the document with an identifier incremented by some predefined value. No extraction rules are need to be provided, and a user only need to define one URL pattern that leads to a document.

When identifiers are strings, the list of available identifiers can be obtained using a search engine of the web resource. A typical search engine provides search results as a list of records that either lead to the desired documents or contain their identifiers. These identifiers (links) can then be collected automatically. The only parameters that user need to define is one extraction rule (to get an identifier (a link) from a search result), and three URL patterns: for a document, the initial search page, and a next page with search results. The Table I summarizes the required user input for both cases.

2) Extraction rules: Extraction rules specify which part of a web document is to be extracted. The format of extraction rules and the way they are set up varies between tools; the rules can be assigned manually or (semi-)automatically using machine learning methods.

In case of manual rule set up, the popular choice is the use of XPath-based expressions, which denote the full path inside the DOM tree of a document to a particular node. XPath notation is especially useful for structured XML documents, as tag names revealing the data being stored; however, for generated HTML documents identification of data by HTML attributes such as `id` and `class` is more common. We propose to use the following notation that simplifies the manual input of extraction rules.

$$\begin{aligned} \mathit{rule} &::= \text{`['}\mathit{token}\text{`]'} \mid \text{`['}\mathit{token}\text{`]'}\ \mathit{rule} \mid \text{`['}\mathit{token}\text{`]'}\ \text{`*'}\ \mathit{rule} \\ \mathit{token} &::= \mathit{name} \mid \mathit{name}\ \text{`,'}\ \mathit{type} \mid \mathit{name}\ \text{`,'}\ \mathit{type}\ \text{`,'}\ \mathit{index} \\ \mathit{type} &::= \text{`id'} \mid \text{`tag'} \mid \text{`cls'} \mid \text{`attr'} \mid \text{`title'} \end{aligned}$$

The *type* specifies the type of DOM element (defaults to 'id'), and *index* specifies its position among siblings (defaults to zero). The star symbol indicates that the token denotes a set rather than a single element; all subsequent tokens are applied to every DOM element in the set.

An extraction rule may be accompanied by a filtering expression that allows omitting unneeded part of the data extracted from a DOM element.

3) Information Extraction Algorithm: Once extraction rules are specified, the web import component fetches bug reports from a remote source in the following way:

1) If numeric identifiers are being used, the identifier of the first document to get is read from settings; otherwise, the list of all available identifiers is obtained by parsing search results.
2) The URL of the next document to fetch is determined by its identifier.
3) The web page with the given URL is parsed, and fields of a structure in the uniform document format are filled according to the extraction rules of the current module.
4) The structure holding the document is stored in the database.
5) The process is repeated until all available documents are processed, or a user-defined limit is reached.

C. Storage

Documents obtained from different web resources are stored in the same database using the uniform format. This allows constructing datasets using arbitrary subsets of documents from multiple sources. Documents from different sources can be selected for inclusion in a dataset based on a module identifier, date range, or metadata.

To simplify initial configuration, Kayur uses the embedded Derby[1] database, which is initialized on the first usage. The tool can be configured to use another database, such as MySQL, instead; the tables to store data will be generated automatically (provided that the database user specified in the settings has all necessary privileges).

D. Text processing

The text processing component performs a sequence of operations on the textual data of selected documents to remove irrelevant information and improve recognition of similar documents. Each operation is optional and customizable. The complete processing sequence includes:

1) Initial filtering.
2) Sentence detection.
3) Tokenization and conversion to lower case.
4) Part-of-Speech (word class) tagging and application of stemming routines.
5) Word filtering.
6) Stop-word removal.

The core operations of tokenization, sentence detection, and Part-of-Speech tagging are performed using OpenNLP [2]. This library relies on binary model files for English language that can be obtained from the website of the OpenNLP project [16].

Kayur includes two types of customizable filters in the format of Java regular expressions. The *initial filters* are applied to the whole text of a document to replace or remove structural elements (such as "Bug description:", "Steps to reproduce:" in bug reports) and non-alphanumeric symbols. The *word filters* do not have replacement functionality, and simply remove words that match a predefined pattern. They are especially useful to remove measurement units, hexadecimal numbers, hyperlinks, and file names.

The component includes custom stemming routines that are applied only to tokens that are detected by Part-of-Speech tagger as:

- Noun, plural.
- Verb: third person singular present, gerund, present participle, past tense, or past participle.

The stemming is performed accordingly to English grammar rules; irregular verbs are converted to dictionary form by the (customizable) list of such forms. Because the tagger can mistakenly detect other parts of speech as verbs, the converted word is validated against the large list of all possible verbs from `libmind` library [17]. If the result is not found in the list, the conversion for this word is skipped.

The last step of text preprocessing operation is stop word removal. Stop words are common words such as articles or pronouns. The tool uses the initial stop word list from MALLET project [18]. The list can be further expanded to include high-frequency words of no particular importance (e. g., "bug", "problem", or "issue").

The tool builds a cache to speed up text processing. Each time a new word is processed, the cache stores the mapping between the word and its final form (or an empty string in case when the word is to be removed).

Once all text processing steps are finished, Kayur saves a corpus structure that contains processed documents, and displays a statistics window that presents a short summary of the data. The summary includes pie charts for metadata, the list of top keywords in the corpus, and length distribution of processed documents.

E. Export

The export component uses a corpus structure prepared during the text processing stage to create an output file in the selected format with a specified term weighting method. Kayur supports two integer weighting methods: Boolean model and raw frequency, and two floating point weighting methods: term frequency and TF-IDF. Kayur supports the following output formats:

1) Attribute-relation file format (ARFF) of WEKA.
2) Plain text format of LDA/HDP topic modeling tools by J. Chang and C. Wang (only integer weights).
3) XML format of Carrot2 (only integer weights).[2]

A user can also set a *rare term threshold* to exclude words that occur in less than a specified number of documents in the corpus. This is important for all text mining tasks, as it greatly reduces the dimension of the resulting vector space, increasing the performance of a data mining tool and reducing resource usage.

[1] https://db.apache.org/derby/

[2] Vector space model is not used when exporting to Carrot XML format, because Carrot2 works directly with textual data. The input file is created by composing documents from processed words.

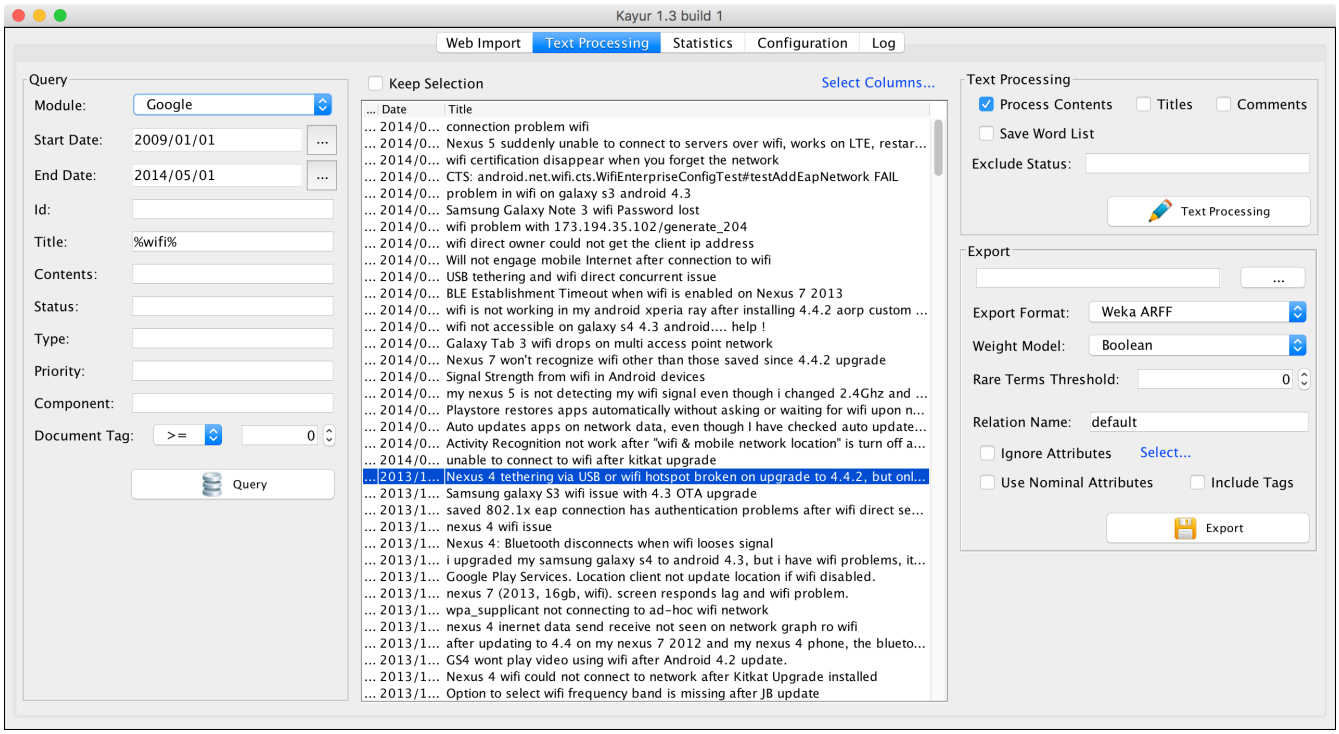

Figure 2. Screenshot of Kayur (Text Processing View)

F. User interface and configuration

Kayur supports both a graphical user interface (GUI) and a command line interface (CLI) designed as an interactive shell (read-eval-print loop). The capabilities of both interfaces are the same, except that the latter supports scripts comprised of valid Kayur commands.

The GUI consists of a single window with five tabs for: 1) the web import component, 2) the text processing component, 3) statistics, 4) program settings, and 5) the journal that contains log messages for the current session (see Figure 2). The GUI is especially useful for initial calibration, as it allows a user to quickly change and test different combinations of parameters.

Although the tool is preconfigured for typical usage scenarios, the GUI allows tuning almost every step of the processing chain to suit particular needs. Web import modules can be freely adjusted to produce desired results. The default internal Derby database can be replaced with a stand-alone database such as MySQL or PostgreSQL. Text processing routines can be disabled or their behavior can be changed by using an external library instead (a call to a library method must be wrapped by a class that implements the interface provided by the tool).

IV. Case Studies and Evaluation

We show two examples of using Kayur, which highlight its abilities to obtain bug report data from various sources using highly customizable filters.

A. Example: Bug Tracking Systems

Bug reports are invaluable source of important information for software developers. Beyond a manual analysis of bug reports, there is also a need to process sets of bug reports as a whole using text mining techniques, such as clustering, to discover trends in software flaws.

Bug reports are usually available on web pages generated by bug tracking systems (BTS). While most of BTS support export of the data into structured formats (XML, JSON), there are also resources that offer the HTML format only.

Kayur allows to generate a dataset from both structured and semi-structured web documents equally easily by providing only few extraction rules. Moreover, as BTS often use numeric identifiers for bug reports, the navigation rules can be omitted.

For example, the settings shown in the Table II are sufficient to generate a dataset from the Gnome bug tracker[3] based on a popular BTS Bugzilla. The rules use the fact that this BTS allows to export a bug report as a structured XML document.[4] A similar simple configuration, shown in Table III, is enough to fetch bug reports from Google Android Issue Tracker, which does not support export in a structured format.[5]

[3]https://bugzilla.gnome.org/
[4]https://bugzilla.gnome.org/show_bug.cgi?ctype=xml&id=[id]
[5]https://code.google.com/p/android/issues/detail?id=[id]

Table II
EXTRACTION RULES FOR GNOME BUGZILLA

Data Field	Extraction Rule
Title	`[short_desc,tag]`
Content	`[long_desc,tag][thetext,tag]`
Comments	`[long_desc,tag]*[thetext,tag]`
Date	`[creation_ts,tag]`

Table III
EXTRACTION RULES FOR ANDROID ISSUE TRACKER

Data Field	Extraction Rule
Title	`[_,title]`
Content	`[meta-container][pre,tag]`
Comments	`[issuecomment,cls]`
Date	`[date,cls][title,attr]`

B. Example: Opinion Mining

Another popular topic that involves web mining is automatic analysis of user opinions. Such analysis is useful, for example, for companies to understand how their products and services are perceived [19]. Kayur makes it easy to create a dataset from user opinions, as they can be stored as comments in the uniform format. As an example, Table IV shows extraction and navigation rules to build a dataset from customer reviews on TV sets available at Amazon.

Table IV
EXTRACTION RULES FOR CUSTOMER REVIEWS AT AMAZON

Data Field	Rule
Title	`[productTitle]`
Content	`[productDescriptionWrapper,cls]`
Comments	`[revMHRL][a-section,cls]*` `[a-spacing-small,cls][a-section,cls]`

C. Performace Evaluation

The performance of the information extraction component is mainly determined by network speed and bandwidth, and by the responsiveness of web resources. For responsive resources, such as Google Android Issue Tracker, and the network speed of 50 Mbps, the document preprocessing rate varies between 2 and 3.5 documents per second. The rate is significantly lower for resources that forbid automatic data fetching, because the delay up to few seconds must be set between subsequent accesses.

The performance of the text processing component on a test machine[6], with all processing steps enabled and default OpenNLP library plugin, is 50 – 60 documents per second. On average, the use of cache reduces the execution time by about 20 %. The main performance factor is the NLP plugin in use, which can be replaced if for a particular task the processing speed is not satisfactory.

[6]Intel Core i5-3210M CPU 2.50 GHz, 4.0 Gb RAM, Windows 7 32-bit

V. RELATED WORK

TraceLab [20] is a highly customizable general-purpose framework for setting up experiments in the form of a data processing tool chain composed of components that are either built-in or created by a user. Compared to Kayur, TraceLab offers richer text processing and visualization capabilities and is more flexible, as it allows to arrange the components in arbitrary way to obtain desired data flow. Although the functionality of TraceLab and Kayur overlap on the text processing stage, Kayur offers the following benefits: 1) uniform access to any web resources that store documents in HTML/XML formats; 2) a persistent storage for fetched documents; 3) a simpler user interface with predefined settings that minimizes time and effort for preparing a dataset from an arbitrary web resource, even if it does not provide API to get data in a structured format; 4) support of popular data mining workbench formats.

There is a variety of stand-alone general-purpose tools to perform subtasks corresponding to those of Kayur's components, but without being tailored for processing of bug reports. Obtaining data from web pages can be performed by Apache Nutch [21]. Text processing can be performed by tools based on powerful toolkits such as NLTK [22] and GATE [23], or even directly in several data mining workbenches, including Carrot2 that implements tokenization, stemming, and stop-word and rare-term removal. Input files for WEKA and Carrot2 can be generated by conversion utilities from other formats such as CSV or XML. The mentioned tools can be arranged to work together to generate results similar to Kayur's, but this can be time-consuming, especially when dealing with multiple sources.

VI. CONCLUSION AND FUTURE WORK

We developed our tool Kayur to speed up laborious fetching and preprocessing steps that are often necessary for raw data obtained from web resources before they can be used in data mining workbenches. The tool is aimed at a broad audience of data mining researchers, as it allows them to obtain real-world data sets relatively easily. It also can be useful for software maintainers that wish to analyze bug reports or user feedback using text mining. As far as we know, Kayur is the only tool that spans the whole sequence of steps needed for textual data processing, ranging from retrieving data from semi-structured documents, over processing it, to exporting it to data mining tools.

For future work, we plan to make extraction rule input easier by providing means for semi-automatic rule generation. We are also working on extending the range of available export formats to include Orange's input formats [24] and the CSV format supported by R [25], and including more term weight methods (such as ConfWeight [26]). Another goal is to provide more ready templates for bug tracking systems and other resources, and examples of plugin usage to better demonstrate the functionality of the tool. We hope

that Kayur's modular design and its scripting interface will inspire novel uses and extensions by its users as well.

REFERENCES

[1] (2016) Kayur. [Online]. Available: http://kayur.net

[2] (2015) Apache OpenNLP. [Online]. Available: https://opennlp.apache.org/index.html

[3] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[4] (2015) Weka 3: Data mining software in Java. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

[5] (2015) Carrot2 – open source search results clustering engine. [Online]. Available: http://project.carrot2.org/

[6] (2015) Topic modeling – Princeton university. [Online]. Available: https://www.cs.princeton.edu/~blei/topicmodeling.html

[7] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[8] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, no. 10, pp. 1411–1428, Oct. 2006.

[9] T. Xie, S. Shi, F. Quan, and C. Yuan, "Research on complex structure-oriented accurate web information extraction rules," in *Proceedings of the 2010 IEEE International Conference on Progress in Informatics and Computing*. IEEE, 2010, pp. 312–316.

[10] G. Shi and Y. Kong, "Advances in theories and applications of text mining," in *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*, ser. ICISE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 4167–4170.

[11] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: A text-based approach to classify change requests," in *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, ser. CASCON '08. New York, NY, USA: ACM, 2008, pp. 23:304–23:318.

[12] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," in *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 311–320.

[13] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 499–510.

[14] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering*, ser. CSMR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 249–258.

[15] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[16] (2015) OpenNLP tools models. [Online]. Available: http://opennlp.sourceforge.net/models-1.5/

[17] (2015) Neuromancer libmind library repository. [Online]. Available: https://github.com/neuromancer/libmind/blob/master/data/pos/verbs/

[18] (2015) Machine learning for language toolkit. [Online]. Available: http://mallet.cs.umass.edu/index.php

[19] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Found. Trends Inf. Retr.*, vol. 2, no. 1-2, pp. 1–135, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1561/1500000011

[20] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes, A. Dekhtyar, D. Manukian, S. Hossein, and D. Hearn, "Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1375–1378.

[21] (2015) Apache nutch. [Online]. Available: http://nutch.apache.org/

[22] E. Loper and S. Bird, "NLTK: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 63–70.

[23] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and Y. Wilks, "Experience of using GATE for NLP R&D," in *Proceedings of the COLING-2000 Workshop on Using Toolsets and Architectures To Build NLP Systems*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, pp. 1–8.

[24] (2015) Orange – loading and saving data. [Online]. Available: http://docs.orange.biolab.si/reference/rst/Orange.data.formats.html

[25] (2016) The R project for statistical computing. [Online]. Available: https://www.r-project.org/

[26] P. Soucy and G. W. Mineau, "Beyond tfidf weighting for text categorization in the vector space model," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pp. 1130–1135.