Published by the IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720

IEEE Computer Society Order Number P5978
ISBN 978-1-5090-6143-3
BMS Part Number CFP16E01-PRT

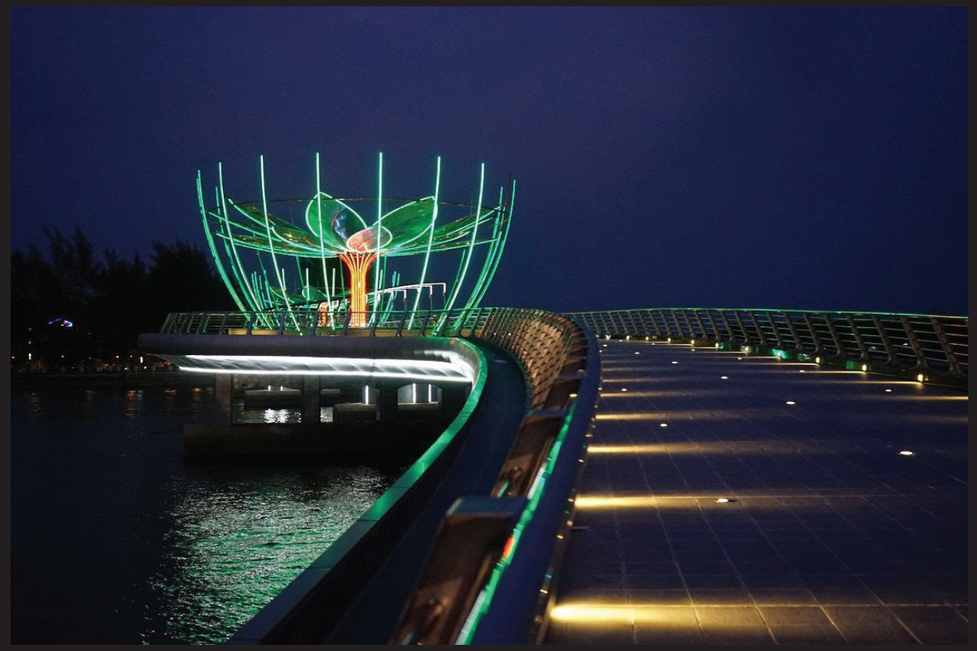
2016 International Conference on Advanced Computing and Applications (ACOMP 2016)

ACOMP 2016

2016 International Conference on Advanced Computing and Applications

23-25 November 2016, Can Tho City, Vietnam

Edited by Lam-Son Lê, Tran Khanh Dang, Josef Küng, Nam Thoai, and Roland Wagner



Proceedings

2016 International Conference on Advanced Computing and Applications

ACOMP 2016

23–25 November 2016
Can Tho City, Vietnam

Editors
Lam-Son Lê
Tran Khanh Dang
Josef Küng
Nam Thoai
Roland Wagner

IEEE Computer Society Order Number P5978
ISBN-13: 978-1-5090-6143-3
BMS Part # CFP16E01-PRT

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Lisa O'Conner
Cover art production by Mark Bartosik

IEEE Computer Society
***Conference Publishing Services* (CPS)**
http://www.computer.org/cps

2016 International Conference on Advanced Computing and Applications

ACOMP 2016

Table of Contents

Section 3: Embedded and Electronic Systems

Section 4: High Performance and Scientific Computing

Section 5: Image Processing and Visualization

Organizing Committee

General Chairs

Thoai Nam, *HCMC University of Technology, Vietnam*
Roland Wagner, *Johannes Kepler University Linz, Austria*

Steering Committee

Elisa Bertino, *Purdue University, USA*
Kazuhiko Hamamoto, *Tokai University, Japan*
Koichiro Ishibashi, *The University of Electro-Communications, Japan*
M-Tahar Kechadi, *University College Dublin, Ireland*
Dieter Kranzlmüller, *Ludwig Maximilians University, Germany*
Josef Küng, *Johannes Kepler University Linz, Austria*
Manuel Clavel, *The IMDEA Software Institute, Spain*
Fabio Massacci, *University of Trento, Italy*
Atsuko Miyaji, *Japan Advanced Institute of Science and Technology, Japan*
Benjamin Nguyen, *Institut National des Sciences Appliqués Centre Val de Loire, France*
Beng Chin Ooi, *National University of Singapore, Singapore*
A Min Tjoa, *Technical University of Vienna, Austria*
Shigeki Yamada, *National Institute of Informatics, Japan*

Program Chairs

Josef Küng, *Johannes Kepler University Linz, Austria*
Tran Khanh Dang, *HCMC University of Technology, Vietnam*
Lam-Son Lê, *HCMC University of Technology, Vietnam*

Publicity Chairs

Phan Trong Nhan, *HCMC University of Technology, Vietnam*
Tran Ngoc Thinh, *HCMC University of Technology, Vietnam*
Le Thanh Sach, *HCMC University of Technology, Vietnam*
Quan Thanh Tho, *HCMC University of Technology, Vietnam*
Hoang Tam Vo, *IBM Research, Australia*

Program Committee

Nguyen Thanh Binh, *HCMC University of Technology, Vietnam*
Stephane Bressan, *National University of Singapore, Singapore*
Michael S. Brown, *National University of Singapore, Singapore*
Pham Quoc Cuong, *HCMC University of Technology, Vietnam*
Dirk Draheim, *Tallinn Technology University, Estonia*
Nguyen Tuan Dang, *University of Information Technology, Vietnam*
Alberto M. Gambaruto, *Barcelona Supercomputing Center, Spain*
Uyen-Synh Ha-Viet, *HCMC International University, Vietnam*
Tran Van Hoai, *HCMC University of Technology, Vietnam*
Trung-Hieu Huynh, *Industrial University of Ho Chi Minh City, Vietnam*
Tomohiko Igasaki, *Kumamoto University, Japan*
Koichiro Ishibashi, *The University of Electro-Communications, Japan*
Ottar Johnsen, *Applied Science University, Switzerland*
Eiji Kamioka, *Shibaura Institute of Technology, Japan*
Surin Kittitornkun, *King Mongkut's Institute of Technology Ladkrabang, Thailand*
Pierre Kuonen, *Applied Science University, Switzerland*
Quan Le-Trung, *Univerisity of Information Technology, Vietnam*
Nanjangud Narendra, *IBM Research, India*
Chin Wei Ngan, *National University of Singapore, Singapore*
Van Vu Nguyen, *HCMC University of Science, Vietnam*
Benjamin Nguyen, *INRIA Rocquencourt, France*
Minh-Son Nguyen, *University of Information Technology, Vietnam*
Thanh-Binh Nguyen, *Da Nang University, Vietnam*
Alex Norta, *Tallinn Technology University, Estonia*
Mizuhito Ogawa, *Japan Advanced Institute of Science and Technology, Japan*
Masato Oguchi, *Ochanomizu University, Japan*
Eric Pardede, *La Trobe University, Australia*
Cong-Duc Pham, *University of Pau, France*
Cong-Kha Pham, *The University of Electro-Communications, Japan*
Tran Minh Quang, *HCMC University of Technology, Vietnam*
Thanh-Sach Le, *HCMC University of Technology, Vietnam*
Thanh-Tho Quan, *HCMC University of Technology, Vietnam*
Shigenori Tomiyama, *Tokai University, Japan*
Tri Kurniawan, *Brawijaya University, Indonesia*
Thai-Nghe Nguyen, *Can Tho University, Vietnam*
Thanh-Nghi Do, *Can Tho University, Vietnam*
Minh-Triet Tran, *HCMC University of Science, Vietnam*
Tuan-Anh Truong, *HCMC University of Technology, Vietnam*
Tran Ngoc Thinh, *HCMC University of Technology, Vietnam*
Nhan-The Luong, *HCMC University of Technology, Vietnam*
Tran-Vu Pham, *HCMC University of Technology, Vietnam*
Hong Trang Le, *HCMC University of Technology, Vietnam*
Pham Hoang Anh, *HCMC University of Technology, Vietnam*

Ngoc-Chau Vo, *HCMC University of Technology, Vietnam*
Duc-Dung Nguyen, *HCMC University of Technology, Vietnam*
Khuong Nguyen-An, *HCMC University of Technology, Vietnam*
Minh-Sang Tran-Le, *HCMC University of Technology, Vietnam*
Phan Trong Nhan, *HCMC University of Technology, Vietnam*
Quang-Minh Huynh, *HCMC University of Technology, Vietnam*
Dinh-Tuyen Nguyen, *HCMC University of Technology, Vietnam*
Lam-Son Lê, *HCMC University of Technology, Vietnam*
Alain Wegmann, *École Polytechnique Fédérale de Lausanne, Switzerland*

Software Keyphrase Extraction with Domain-specific Features

Oscar Karnalim
Faculty of Information Technology
Maranatha Christian University
Bandung, Indonesia
oscar.karnalim@gmail.com

Abstract— **Despite the fact that keyphrase is widely used as a brief summary to represent documents, most keyphrase extraction is only focused on arbitrary text. However, many document types have specific behavior which require particular pre-processing in order to extract keyphrases. In software domain, keyphrases can only be extracted by utilizing reverse-engineering approach and applying several conversion rules. This paper proposes a mechanism to extract software keyphrases with domain-specific features. For our case study, our proposed method is applied to Java Archive, a distributional form of Java binaries. Besides pre-processing and conversion rules, our method also utilizes the combination of supervised and unsupervised keyphrase extraction approach to exploit the benefits of both approaches. Furthermore, in order to extract keyphrase pattern more accurately, software-related features are also incorporated besides standard keyphrase extraction features. These features are software structure, software-related natural language text, and software term association. Based on overall evaluation, our proposed method yields moderate R-precision. Thus, our approach is quite considerable to be applied for extracting software keyphrase.**

Keywords—keyphrase extraction; software; domain-specific features; Java Archive

I. INTRODUCTION

Keyphrase is widely used as a brief summary to represent documents [1]. They help readers rapidly understand document context without the need to know the entire document. In software domain, keyphrase is often featured in software portal such as Maven repository [2] and NuGet [3]. Each software is tagged with one or more keyphrase(s) in order to provide prior knowledge about the software. For example, ANTLR will be tagged by "compiler" and "parser" keyphrases due to its functionality. However, keyphrase is only found when a software is listed on a particular software portal. In fact, most software is not listed on software portal and determining keyphrase for these software may be a tedious task. Human tagger is required to know the software subject even though software content cannot be read directly. Thus, automatic keyphrase extraction for software domain is highly desirable.

There has been various works about keyphrase extraction which primarily focused on arbitrary text. Most of them can be classified into two approaches: supervised and unsupervised approach [4, 5]. In supervised approach, each keyphrase candidate is fed to a trained classifier and resulted keyphrases are fully-determined by classification algorithms. On the other hand, unsupervised approach ranks keyphrase candidates based on their importance and N candidates with the highest score will be selected as keyphrases. Based on their respective mechanism, supervised approach is more beneficial to be conducted when keyphrase characteristics are still hidden. However, since classification result are not comparable, unsupervised approach is more desirable when the resulted keyphrases are limited and should be the most relevant ones. Determining the most relevant keyphrases among all keyphrases requires comparison between each keyphrase based on its relevancy.

Since both approaches have their unique benefits which complements to each other, this paper proposes a combination of supervised and unsupervised approach for extracting software keyphrases. Keyphrase importance is still comparable while hidden keyphrase pattern can be detected automatically through learning mechanism. Several software-specific features are also involved to improve the effectiveness of our proposed method. These features are software structure, software-related natural language text, and software term association. Furthermore, since software content cannot be read directly, an additional pre-processing for converting binary format to readable well-formed sentences is also proposed. For our initial step, pre-processing is focused on Java Archive, a distributional form of Java binaries that can be used either as stand-alone program or library. Yet, our approach can also be applied to other software type as long as it yields similar extracted features.

II. RELATED WORKS

Keyphrase extraction is a task to determine which phrases are the most important and the most representative from document body [6]. This task is typically split into two phases which are selecting keyphrase candidates and determining correct keyphrases [5]. Selecting keyphrase candidates is conducted using several heuristic rules such as: 1) removing stop words; 2) allowing only phrases with particular part-of-speech tags; 3) extracting n-grams, and 4) filtering only high-frequency phrases. On the other hand, determining correct keyphrases can be conducted using either supervised or unsupervised approach. Supervised approach relies on a trained classifier to determine keyphrases whereas unsupervised approach selects the most important candidates as keyphrases based on several assumptions.

978-1-5090-6144-0/16 $31.00 © 2016 IEEE
DOI 10.1109/ACOMP.2016.9

Various learning algorithm has been conducted in supervised approach such as naive bayes [1, 7], decision tree [6], neural network [8], and support vector machines [9]. However, since learning algorithm is heavily dependent with instance features, proper instance features are more crucial to be considered than the learning algorithm itself. Even the best learning algorithm will yield low accuracy if its instance features are not properly defined. Instance features are commonly classified into four categories which are: statistical, structural, syntactic, and external resource-based features. Among these features, external resource-based features are the only features that involve resource other than training data.

On the contrary, unsupervised approach consider keyphrase extraction as a ranking problem instead of classification task [4]. Keyphrase candidates are ranked based on their importance and Top-N candidates will be selected as keyphrases. Yet, the definition of importance varies based on several assumptions. This includes classic IR ranking (e.g. TFIDF), graph-based ranking [10], topic-based clustering [11], simultaneous learning [12], and language modeling [13].

In this paper, supervised and unsupervised approach are combined so that keyphrase importance is comparable to each other while hidden keyphrase pattern still can be learned automatically. This combination is conducted since software keyphrase characteristics may be vary between software type but its result should still be limited at a certain number. All keyphrase candidates are extracted from software using reverse-engineering approach, selected with particular heuristics, queued based on their importance, and fed to a classifier until N keyphrases are classified. Furthermore, several software-specific features are implicitly involved on these phases: Software structure is utilized on pre-processing and candidate classification; Software-related natural language text is utilized on candidate selection, ranking, and classification; and software term association is utilized for candidate ranking. The two latter features are based on noun phrases extracted from 14.433 GitHub html files. These html files are scraped from 16.000 links at the beginning of GitHub Java Corpus project list [14] where remained 1.567 links are not accessible. For convenience, these 14.433 GitHub html files are referred as GitHub html files on the rest of this paper.

III. METHODOLOGY

Proposed software keyphrase extraction consists of four phases: 1) document body extraction; 2) keyphrase candidate selection, 3) keyphrase ranking; and 4) keyphrase classification. Among these phases, document body extraction is the only phase that is highly-related with software structure. This phase should be modified when target software structure is changed or extended. Moreover, the combination of supervised and unsupervised approach is conducted on the last two phases. Unsupervised approach is conducted on keyphrase ranking whereas the supervised one is conducted on keyphrase classification. For each software, our proposed approach is expected to extract the most representative keyphrases based on software functionality. For example, extracted keyphrases from MySQL connector should be “database” and “SQL driver”. Both terms are relevant with MySQL connector functionality which aim to manipulate SQL database.

A. Document Body Extraction

This phase is responsible to extract all textual information from target software and convert it into sentences. Yet, since extraction can only be conducted when software structure is known, our approach is focused on Java Archive structure as a case study. For each Java Archive (JAR), our approach extracts textual information from five major parts: package name, class name, field name, method name, and string literal in method body (which is responsible for all text in program contents). Each extracted text will be tokenized based on the combination of natural language delimiter and Java naming rules. This tokenization is adapted from previous research about JAR search engine [15].

However, several additional rules are also applied in order to form well-formed sentences from the extracted text. For each method name, a dummy token “I” will be concatenated at the beginning of extracted tokens. This rule is conducted based on an assumption that method name is always started with a verb and a well-formed sentence is typically started with a noun. Thus, for each string literal in method body, all string literals collected within a method body will be concatenated as a paragraph before tokenized where each newline will be replaced with a full stop mark (.). This rule is associated with two assumptions which are: 1) Sentences in method body is commonly split into several string literals due to programming behavior and variable usage; 2) Full-stop mark is commonly replaced with newline in string literal that represents software output.

B. Keyphrase Candidate Selection

In order to reduce the workload of keyphrase ranking and classification, not all phrases are taken as keyphrase candidates. Keyphrase candidates are selected with several heuristics which are:

a) Keyphrase candidate should be a noun phrase with phrase length lower or equal with 4 words. Noun phrase identification is adopted from Sarkar et al [8] with an additional rule to treat undefined token and foreign word as a noun. Proposed regular expression for noun phrase identification can be seen in (1) which acronyms are based on Penn Treebank part-of-speech notation, Part-of-speech of each token is determined with Stamford log-linear part-of-speech tagger [16].

$$(\varepsilon+DT)(JJ+JJR+JJS)^{*}(NN+NNP+NNS+NNPS)^{+} \quad (1)$$

b) Keyphrase candidate should not contain stop words as its keyphrase member. This heuristic is inspired from Frank et al keyphrase selection [7] but applied in Java Archive instead of arbitrary text. Stop words for Java Archive are taken from our previous research [15].

c) Each token in keyphrase candidate should contain at least 3 characters. This heuristic is inspired from early IR system that removes these kind of terms for indexing [17]. Furthermore, it is also strengthened by our finding in manual observation that short-sized term is seldom occurred on relevant keyphrase.

d) Keyphrase candidate should be listed on software-related natural language text with an assumption that

natural language text only contains well-formed noun phrases. This heuristic is utilized to exclude over-technical identifier names from keyphrase candidate list. Software-related natural language text is taken from GitHub html files.

C. Keyphrase Ranking

In this phase, keyphrase candidates will be ranked based on their importance and stored on a list in descending order. To measure candidate importance, our approach utilizes a scoring function in (2) that is extended from Frank et al TFIDF scoring [7]. TFIDF is selected as a baseline for our ranking function since it offers very robust performance across different dataset [4]. TFIDF in our approach is extended with an assumption that keyphrase should be the most related noun phrase toward other document noun phrases. This rule is applied by summing all relatedness score between the selected candidate with other noun phrases on the given document. *score(t,D)* stands for the score of a noun phrase *t* in document *D*. *tf(t)* represents term frequency of noun phrase *t*, *df(t)* represents document frequency that contain noun phrase *t*, *N* represents the number of documents in collection, and *rel(t,td)* represents the relatedness score of noun phrase *t* and *td*.

$$score(t,D) = tf(t)*(-^{2}log\,(df(t)/N))*\sum_{td \in D} rel(t,td) \quad (2)$$

However, since the semantic relation in software domain may be different with standard natural language domain, our approach incorporates software-related natural language text as a basis of our relatedness measurement. Relatedness is measured with asymmetric noun phrase association extracted from GitHub html files. Asymmetric association is utilized instead of symmetric one based on following reasons: 1) keyphrase should be a noun phrase that have high relatedness degree with all other noun phrases but not necessarily vice versa; and 2) equal-symmetric association is rarely occurred in real-world noun phrase relatedness. Noun phrase relatedness equation can be seen in (3) which is adapted from simple conditional probability. *df(t,td)* is the number of documents that contain noun phrase *t* and *td* whereas *df(t)* is the number of documents that contain noun phrase *t*.

$$rel(t,td) = \frac{df(t,td)}{\mathrm{df(t)}} \quad (3)$$

D. Keyphrase Classification

After ranked and stored on a list, each keyphrase candidate will be popped out from the beginning of the list and fed to a classifier until N keyphrases are selected. Our approach utilizes a classifier called logistic regression that is implemented by WEKA [18]. Logistic regression is a discriminative classifier that learn function P(Y|X) in the case where Y is discrete-valued, and $X = (X_1 \ldots X_{ni})$ is any vector containing discrete or continuous variables [19]. We prefer discriminative classifier to generative one since it tends lower asymptotic error while sufficient training data exist [20]. Moreover, logistic regression is selected due to its simplicity to handle binary-valued target class. As we know, our classification task is only required to classify whether a candidate is a keyphrase or not. Other discriminative approaches may be more inefficient due to their complexity.

When utilizing instance features, our approach incorporates three feature categories suggested by Hasan & Ng [5]. These categories include statistical, structural, and external resource-based feature. According to Hasan & Ng, syntactic feature should be excluded due the fact that it is not useful in the presence of other categories. The detail of proposed instance features can be seen in Table I. Statistical features involved in this research are standard features for keyphrase extraction whereas structural and external resource-based features are software-specific features. Software-specific features are expected to improve the accuracy of keyphrase classification.

TABLE I. INSTANCE FEATURES

ID	Feature	Type
TF	Term frequency	Statistical
IDF	Inverse document frequency	
PDD	Phrase distance in document	
PDS	Phrase distance in sentence	
PL	Phrase length	
WC	Word count	
SP	Software-based structural position	Structural
TFNL	Term frequency in software-related natural language text	External resource-based
IDFNL	Inverse document frequency in software-related natural language text	

Statistical features consist of term frequency (TF), inverse document frequency (IDF), phrase distance in document, phrase distance in sentence, phrase length, and word count. The first three features are inspired from Frank et al features [7] but differ in how TF and IDF are represented. As suggested by Hulth [21], TF and IDF are split as two separate features instead of merged as one. Phrase distance is measured by the number of terms that occur before the first occurrence of the target phrase. However, to avoid misleading pattern, phrase distance is normalized based on their respective location length. Phrase distance in sentence is measured locally within a sentence whereas phrase distance in document is measured globally on a document. Besides occurrence-based statistical features, two lexical-based features are also involved. These features are phrase length and word count. Phrase length represents keyphrase candidate string size whereas word count represents the number of word contained on that keyphrase candidate.

Since structural features are highly related with software domain, our approach utilizes Java Archive structure as a case study. Five major parts which have been described in document body extraction are utilized as structural location. These parts include package name, class name, field name, method name, and string literal in method body. However, instead of storing structural location as a single multi-valued feature, these structures are combined with occurrence-based statistical features. Each occurrence-based statistical feature is split into five sub-features based on its structural location. By conducting this combination, we have 22 features so far (5 structural component * 4 occurrence-based statistical features + 2 lexical statistical features). Lexical statistical features are not

combined with structural location since these features are not affected with software structure.

External resource-based features involved in this approach assume that software keyphrases should commonly occur in software-related natural language text. In our case, software-related natural language text is based on GitHub html files and keyphrase occurrence will be calculated using term frequency and inverse document frequency. With these two additional features, our proposed instance features consist of 24 features in total. These features are expected to be declarative enough for classifying keyphrases.

IV. EVALUATION

A. Evaluation Dataset

Since there is no publicly available dataset that fit our needs, 107 Java Archives (JAR) are collected with Google search engine, annotated, and treated as our dataset. To avoid misleading result, our dataset only consists of JARs that is listed on their own websites. JARs that have their own websites are assumed to be well-developed and following proper software structure and naming rules. Our dataset is split into two categories which are 55 standalone JARs and 52 component-based JARs. Standalone JAR is a JAR that can be utilized without relying on the other JAR whereas component-based JAR can only be utilized with the existence of other JARs.

Relevant keyphrases on each JAR are manually assigned by the author of this paper wherein each relevant keyphrase is selected from JAR terms. This keyphrase assignment yields 485 relevant keyphrases for 107 JARs. Furthermore, assigned keyphrase reliability is also validated with an inter-rater agreement called Fleiss' kappa [22]. Fleiss' kappa is conducted to 7 CS undergraduate students that are quite familiar with software-related terms, keyphrase, and keyword-based searching. These students are asked to check each assigned keyphrases and judge its relevancy with a boolean value (true means relevant and false means irrelevant). Based on Fleiss' kappa statistic of student agreement, our assigned relevant keyphrases are reliable since it yields 89,81% of agreement. Thus, these assigned keyphrases are considered valid to be used in our dataset.

B. Keyphrase Approximate Matching

Despite keyphrase extraction can be evaluated based on how many relevant keyphrases are retrieved by the system, exact match between retrieved and relevant keyphrases is overly strict condition [5]. Several retrieved keyphrases may be related to relevant keyphrases but not exactly in same lexical form (e.g. "Java Archives" and "Java Archive"). Moreover, retrieved keyphrases may be just a sub-phrase or super-phrase of relevant keyphrases (e.g. "Java Archive" and "System-related Java Archive"). These matching problems are caused by linguistic phenomena [23] that commonly occurs in the natural language.

In order to overcome these problems, our proposed method is evaluated with approximate matching instead of exact matching. The details of approximate keyphrase matching procedure can be seen in Figure 1. Generally, this procedure is split into three sub-procedures which are morphological, partial, and normalized symmetric association matching. Morphological matching handles exact match and any morphological variations that can be solved with stemming; Partial matching handles matching problem when retrieved keyphrase is a sub-phrase or super-phrase of relevant keyphrase; and normalized symmetric association matching handles the rest of linguistic phenomena based on mutual information.

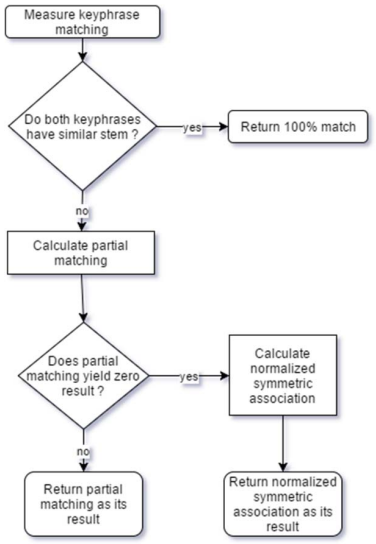


Figure 1. Keyphrase Matching

Partial matching is measured with the longest common sub-phrase (LCS) between retrieved and relevant keyphrase. However, since both keyphrases are noun phrase which main term is typically placed at the end of the phrase, selected LCS should share similar last term with both keyphrases (e.g. LCS "bytecode" shares similar last term with "java bytecode" and "compiler bytecode"). Partial matching between two keyphrases will be measured with (4). *rt* and *rl* are retrieved and relevant keyphrase respectively, *LCS(rt,rl)* is the word count of the longest common sub-phrase shared by both keyphrases, *size(rt)* is the word count of retrieved keyphrase, and *size(rl)* is the word count of relevant keyphrase. The bottom part of the equation is conducted to normalize matching result as percentage based on maximum similarity. This percentage result will represent the matching degree between retrieved and relevant keyphrase.

$$partial\_match(rt,r) = \frac{LCS(rt,rl)}{\min(\text{size(rt)},\text{size(rl)})} \quad (4)$$

Normalized symmetric association matching is calculated based on mutual information, a popular symmetric association measurement [17]. Two keyphrases are strongly associated with each other if they always co-occur in similar documents within natural language corpus. GitHub html files are utilized as our natural language corpus and the result of mutual information is normalized as a percentage in order to represent the degree of keyphrase relatedness. Normalized mutual information utilized in this research can be seen in (5). *df(rt,rl)* is the number of documents that contain retrieved and relevant

keyphrase whereas *df(rt)* and *df(rl)* are the number of documents that contain keyphrase *rt* and *rl* respectively. Multiplication of *df(rt,rl)* on the upper part of this equation is conducted to normalize its result as percentage so that two mutually associated keyphrases will yield 100% match as its result whereas two unassociated keyphrases will yield zero result.

$$normalized\_MI(rt,rl) = \frac{df(rt,rl) * df(rt,rl)}{\mathrm{df(rt)} *, \mathrm{df(rl)}} \quad (5)$$

C. *Evaluation Schema*

The effectiveness of keyphrase extraction is evaluated with R-precision based on given dataset. R-precision calculates the precision based on Top-R retrieved keyphrases for each document (software) wherein *R* represents the number of actual relevant keyphrases. For example, if a software has 3 relevant keyphrases and 2 of them are in Top-3 retrieved keyphrases, its R-precision will be 2/3 = 0.66 (66%). R-precision is selected as evaluation measurement instead of standard precision and recall based on two reasons: 1) The number of relevant keyphrases for each software in our dataset are vary. Thus, calculating precision and recall at a particular threshold may yield biased result; and 2) Keyphrase extraction is commonly focused only on top-ranked keyphrases. Therefore, utilizing larger threshold for calculating precision and recall may also be biased. Keyphrase approximate matching is also incorporated on R-precision to handle linguistic phenomena between retrieved and relevant keyphrases.

Since each keyphrase can only be compared once when calculating R-precision, comparison pairs will be selected based on following steps: 1) all retrieved-relevant pairs are generated based on retrieved and relevant keyphrases for the given software; 2) all pairs are sorted based on its approximate matching in descending order and; 3) each pair which member(s) is already a member of higher approximate matching pair is removed. Afterwards, the remaining pairs will be utilized for calculating R-precision. This selection mechanism assures that each keyphrase can only be compared once on remaining pairs and selected pairs will yield the highest approximate matching degree among all possible comparison pairs.

D. *Evaluating Keyphrase Candidate Selection Heuristics*

Since the main goal of candidate selection is to remove as many as possible keyphrase candidates without reducing overall effectiveness, the number of removed keyphrase candidates should also be considered when evaluating each candidate selection heuristic. A heuristic is considered as a 'good' heuristic *iff* it removes many keyphrase candidates without reducing its overall effectiveness. Furthermore, several potential heuristics beside proposed heuristics are also evaluated in this paper for comparison purpose. These heuristics are based on related works and several assumptions toward our keyphrase extraction. These approaches are excluded from our methodology since they reduce overall effectiveness as a tradeoff of removing many keyphrase candidates.

Evaluation result of these heuristics can be seen in Table II wherein the 'good' heuristics are marked with blue color. These evaluations are conducted under standard TFIDF keyphrase ranking and incorporate all instance features for keyphrase classification. However, due to the fact that most candidate selection restricts its candidates as noun phrases which contain no stop words, the first two heuristics from our approach (noun phrase constraint and no-stop-word constraint) are assumed to be 'good' heuristics and excluded from our evaluation. Instead, these features are utilized as default baseline for evaluating the effectiveness of each heuristic. As seen in Table II, heuristics used in our methodology (H2 and H3) are considered as 'good' heuristics. Both of them remove many keyphrase candidates without lowering R-precision. These heuristics even improve R-precision by removing several false positive keyphrases. This result strengthens the fact that short-sized term is seldom occurred on relevant keyphrase and relevant keyphrase is typically a well-formed noun phrase.

TABLE II. KEYPHRASE CANDIDATE SELECTION HEURISTICS

ID	Candidate Selection Heuristics	R-precision	Removed Keyphrase Candidates
H1	Default	28.945%	0%
H2	Minimum 3 characters per term in phrase	29.999%	10.538%
H3	Exclude phrase that is not listed on software-related natural language text	29.452%	44.907%
H4	Exclude shorter phrase which n-grams are overlapped with longer phrase(s)	13.537%	46.305%
H5	Exclude phrase which occurs only once in document	28.479%	49.812%
H6	Only select phrases which normalized TF is in range of normalized keyphrase TF	28.759%	1.043%
H7	H2 + H3	29.346%	55.185%

H4, H5, and H6 are potential heuristics which are evaluated for comparison purpose. H4 assumes that longer keyphrase may be more preferable as a keyphrase candidate due to its more-specific meaning. However, this heuristic lower R-precision significantly since many relevant keyphrases are not always the most specific phrase on the document. H5 is inspired from Frank et al research [7] which removes all candidates that only occurs once in a document. Despite of its huge amount of removed keyphrase candidates, this heuristic is not suitable on software context. Several keyphrases only occur once in software due to programming concept about code reusability. On the contrary, since this heuristic only reduces a small amount of overall effectiveness as a tradeoff, this heuristic may be potential to be utilized on dataset with many large-sized software. H6 assumes that relevant keyphrases should have a particular TF pattern toward their respective document noun phrases. Keyphrase candidates are restricted to all candidates which TF is in range of keyphrase TF. Yet, normalization is also incorporated for calculating TF to avoid biased result. For example, if a noun phrase "bytecode" occurs 5 times on a document that consists of 10 noun phrases, it can be concluded that normalized TF (NTF) of "bytecode" is 5/10 = 0.5 (50%). The statistic of all NTF from our relevant

keyphrases can be seen in Figure 2. Horizontal axis represents frequency-ordered list of keyphrases whereas vertical axis represents NTF. From this statistic, the boundary of keyphrase NTF can be deduced which are 0.143% as upper bound and 5.846E-06% as lower bound. Unfortunately, many irrelevant keyphrases are also appeared on that boundary range. Thus, H6 have no significant impact toward removed keyphrase candidates. Additionally, H6 also reduces R-precision since NTF of several approximate-matched keyphrases are out of keyphrase range. Based on Figure 2, it can be concluded that keyphrases does not always entailed from most frequent noun phrases. Most keyphrases even have low TF on their respective documents.

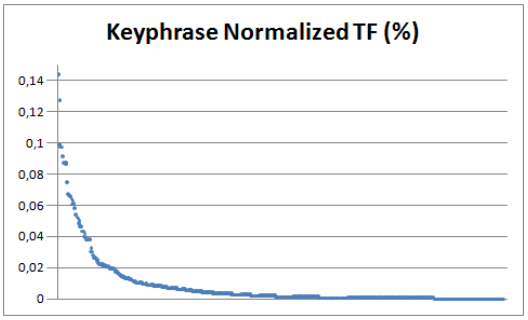


Figure 2. Keyphrase normalized TF based on their respective JAR

H7 is conducted to see the impact of combined 'good' heuristics (H2+H3). Although its R-precision improvement is lower than H2 and H3, this combination is still considerable since this heuristic removes the largest amount of keyphrase candidates among all evaluated heuristics without lowering R-precision.

E. *Evaluating Keyphrase Candidate Ranking*

Keyphrase candidate ranking is evaluated by comparing our proposed ranking with other extended TFIDF ranking functions based on R-precision. Our proposed ranking is considered as a good ranking function *iff* it outperforms other ranking functions. For environment baseline, all ranking functions are conducted with H7 candidate selection heuristic and candidate classification that utilize all instance features. The detail of all evaluated ranking functions can be seen on Table III. R2, R3, and R4 are evaluated for comparison purpose whereas R5 is our selected candidate ranking function. R2-R5 are conducted by utilizing GitHub html files as external resource.

TABLE III. RANKING FUNCTIONS AND ITS ASSUMPTIONS

ID	Ranking Schemes	Assumption
R1	Standard TFIDF	-
R2	TFIDF * External TFIDF	External TFIDF may strengthen the score of natural language keyphrases
R3	TF * External IDF	The replacement of IDF with EIDF may yield more consistent IDF score
R4	TFIDF * the sum of symmetric term association between selected keyphrases with other noun phrases	Keyphrases should be highly-related with other noun phrases in JAR since they aim to describe similar generic functionality of the given JAR
R5	TFIDF * the sum of asymmetric term association between selected keyphrases with other noun phrases	Similar with R4 except that it assumes relatedness between keyphrases and other noun phrase is asymmetric instead of symmetric relation.

R-precision for each ranking function can be seen in Figure 3. External TFIDF is incorporated in R2 by multiplying internal TFIDF with external TFIDF. This approach yields lower R-precision than standard TFIDF since not all keyphrases are guaranteed to have high TFIDF score on both domains (internal and external resource). Most of them only have high internal TFIDF score. R3 assumes that IDF may be more accurate when extracted from software-related natural language instead of the software itself. However, it yields lower R-precision since IDF pattern on both domains are quite different. R4 is quite similar with our approach (R5) except that R4 utilize symmetric term association instead of asymmetric one. R4 incorporates mutual information as its symmetric term association. Yet, since keyphrases should not related to other noun phrases in symmetric manner, this approach still yields lower R-precision than standard TFIDF. R5 is our proposed ranking function which yields the highest overall effectiveness among other ranking schemes. With this fact, our approach is proved to be the most effective approach. Moreover, it can also be concluded that ranking function for software keyphrase extraction should involve relatedness between keyphrase and other noun phrases asymmetrically.

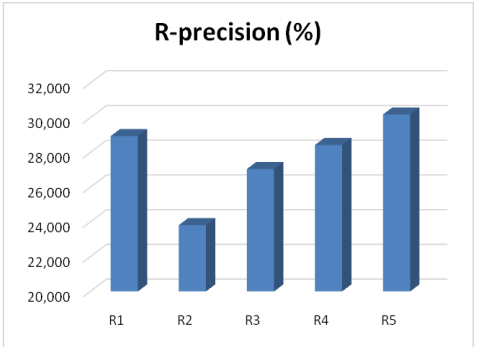


Figure 3. R-precision of all ranking functions

F. *Evaluating Instance Features*

In order to prove the effectiveness of each instance feature, these features are mapped into several schemes and compared with default scheme. A feature is considered as a positive-impact feature *iff* its generated scheme yields higher accuracy than the default one. Generated scheme is the combination of evaluated instance feature with default scheme features. Default scheme consists of two features which are TFIDF and phrase location in document (PDD). These features are adapted from Frank et al features [7] and assumed as positive-impact features due to their frequent entanglement in many keyphrase extraction mechanisms. Thus, instance features evaluated in this section are PDS, PL, WC, SP, and external TFIDF (TFNL + IDFNL). Additionally, the impact of split TFIDF is also evaluated in this section.

Despite most instance features can be mapped into a scheme by just combining selected feature with default features, several features can only be mapped with exclusive mechanism due to its behavior. Generating a scheme for split TFIDF can only be conducted by replacing default TFIDF feature with TF and IDF (*D+TF+IDF-TFIDF*) whereas structural features require default features to be split based on its respective structural position (*D U SP*).

Each scheme is evaluated using 10-fold cross-validation with WEKA based on our keyphrase dataset. However, since the number of irrelevant keyphrases on each software tends to be greatly higher than the number of relevant ones, irrelevant keyphrases are limited to unigram, bigram, trigram, and quartogram noun phrases with the highest TF. This selection mechanism is conducted to gain balanced keyphrase dataset (485 relevant keyphrases and 402 irrelevant keyphrases).

Evaluation result of each scheme can be seen in Figure 4. *D* represent default scheme, *All* represents a scheme that include all features, and the other acronyms represent instance feature which detail can be seen in Table I. Since all non-default schemes yield higher accuracy than default scheme, it can be concluded that all instance features involved on our learning model are positive-impact features. This conclusion is also strengthened with the fact that combining all instance features yields the highest accuracy rate (83.99%). Keyphrase word count (WC) is the most impactful features because most keyphrases consists of two or three words whereas phrase distance in sentence (PDS) is the least impactful ones since its pattern is quite similar with default scheme's phrase distance in document (PDD).

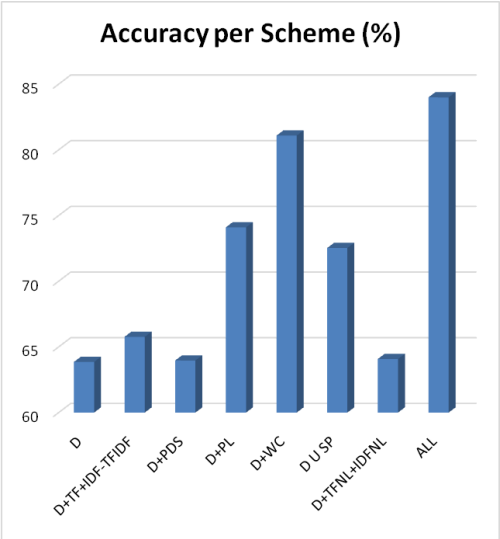


Figure 4. Evaluation Result per Scheme

Besides evaluating each instance feature separately, the impact of each instance feature for overall classification task is also evaluated through logistic regression odd ratio. Odd ratio for each feature on a particular target class indicates the impact of that feature for classifying that target class. Odd ratios for "keyphrase" target class can be seen in Figure 5. "keyphrase" target class is greatly influenced with PDS from string literal in method body (string_literal_in_method_PDS) since it yields the highest odd ratio (3,4996). This result yields the fact that most keyphrase can be found on string literal in method body with a particular location pattern. On the other hand, WC yield the lowest odd ratio for classifying "keyphrase" target class (0,0229). Yet, since our classification task only involves two target class, it can be concluded that WC is the most impactful feature for determining the other target class ("not keyphrase" target class) as its output. This result also strengthens the fact that most keyphrases consists of two or three words.

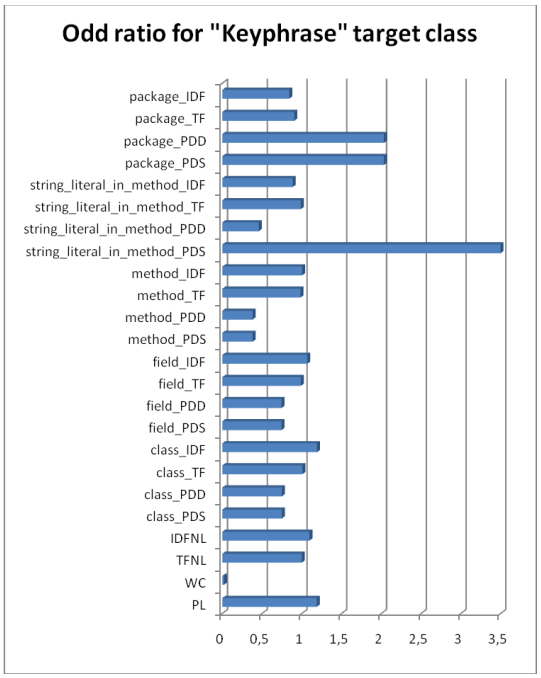


Figure 5. Odd ratio for "Keyphrase" target class

G. *Evaluating Dataset Characteristic*

Since there are two kind of JAR in our dataset, the impact of each JAR type is also evaluated under our proposed method. For this purpose, three schemes are generated which involving the whole dataset, only standalone JARs, and only component-based JARs. Evaluation result of these schemes can be seen in Figure 6. Scheme that involving only standalone JARs yields the lowest R-precision (29.118%) whereas scheme that involving only component-based JARs yields the highest one (31.942%). This phenomenon is caused by the number of keyphrase candidates for each dataset. Standalone JAR tends to have more keyphrase candidates than component-based JAR due to its larger size. This statement is consistent with Hasan & Ng research that states the difficulty of keyphrase extraction is increased proportionally with document length due to candidate size [5]. When evaluated on the whole dataset, our proposed method yields moderate R-precision which is 31.556%. This precision is quite acceptable when compared with other keyphrase extraction evaluation result [5].

V. CONCLUSIONS

In this paper, a software keyphrase extraction has been

developed. Textual information from software are extracted using reverse-engineering approach, selected with several heuristics, ranked with extended TFIDF, and classified as keyphrase based on logistic regression. We have evaluated heuristics used in candidate selection, ranking function in candidate ranking, and instance features in candidate classification. All of them are proved to be effective based on our evaluation. Additionally, software-related features incorporated on our proposed method are also considered to be quite effective despite of its moderate impact. From dataset perspective, our software keyphrase extraction works well on small-size software due to its limited keyphrase candidate. However, our approach is still considerable to be applied on large-size software since large-size software dataset only reduces a small amount of R-precision.

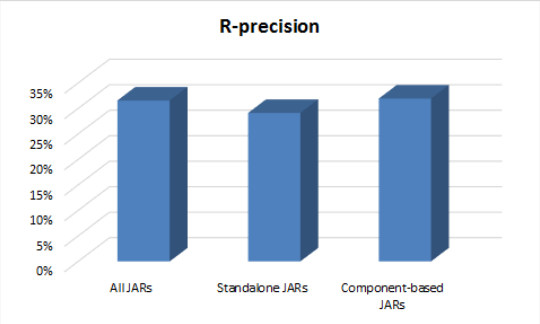


Figure 6. R-precision based on Dataset

VI. FUTURE WORK

In next research, we will exploit more software textual information by utilizing supplementary files that usually co-exist with software (e.g. textual databases, configuration file, and source code). These files should consist of several features that may strengthen the keyphraseness of relevant keyphrases. Additionally, we also intend to evaluate our approach on various software files other than JAR (e.g. Windows executable files). From system perspective, keyphrase extraction proposed in this research will be incorporated to our Java Archive search engine [15, 24, 25] so that each JAR search result will be featured with relevant keyphrases. These keyphrases are expected to give a generic description about the given JAR so that user may choose their relevant JAR more easily.

REFERENCES

[1] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin and C. G. Nevill-Manning, "KEA: Practical automatic keyphrase extraction," in *The fourth ACM conference on Digital libraries*, 1999.

[2] "Maven Repository," Apache, 2006. [Online]. Available: http://mvnrepository.com/. [Accessed 28 2 2016].

[3] "NuGet Gallery," .NET Foundation, 2016. [Online]. Available: https://www.nuget.org/.

[4] K. S. Hasan and V. Ng, "Conundrums in unsupervised keyphrase extraction: making sense of the state-of-the-art," in *The 23nd International Conference of Computational Linguistics*, 2010.

[5] K. S. Hasan and V. Ng, "Automatic Keyphrase Extraction: A Survey of the State of the Art," in *The 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[6] P. Turney, " Learning Algorithms for Keyphrase Extraction," *Information Retrieval,* vol. 2, no. 4, pp. 303-336, 2000.

[7] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin and C. G. Nevill-Manning, "Domain-specific Keyphrase Extraction," in *16th International Joint Conference on Artificial Intelligence*, 1999.

[8] K. Sarkar, M. Nasipuri and S. Ghose, "A New Approach to Keyphrase Extraction Using Neural Nework," *International Journal of Computer Science,* vol. 7, no. 2, 2010.

[9] P. Lopez and L. Romary, "Automatic key term extraction from scientific article in GROBID," in *5th International Workshop on Semantic Evaluation*, 2010.

[10] R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in *20014 Conference on Empirical Methods in Natural Language Processing*, 2004.

[11] L. Zhiyuan, W. Huang, Y. Zheng and M. Sun, "Automatic Keyphrase Extraction via Topic Decomposition," in *Empirical Methods in Natural Language Processing*, 2010.

[12] X. Wan, J. Yang and J. Xiao, "Towards an iterative Reinforcement Approach for Simultaneous Document Summarization dan Keyword Extraction," in *45th Annual Meeting of the Association of Computational Linguistics*, 2007.

[13] T. Tomokiyo and M. Hurst, "A Language Model Approach to Keyphrase Extraction," in *The ACL Workshop on Multiword Expressions*, 2003.

[14] M. Allamanis and C. Sutton, "Mining Source Code Repositories at Massive Scale using Language Modeling," in *The 10th Working Conference on Mining Software Repositories*, 2013.

[15] O. Karnalim, "Java Archives Search Engine Using Byte Code as Information Source," in *International Conference on Data and Software Engineering (ICODSE)*, Bandung, 2014.

[16] K. Toutanova, D. Klein, C. Manning and Y. Singer, "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network," in *The North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2003.

[17] B. Croft, D. Metzler and T. Strohman, Search Engine : Information Retrieval in Practice, Boston: Pearson Education .Inc, 2010.

[18] M. Hall, E. Frank, Holmes Geoffrey, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations,* vol. 11, no. 1, 2009.

[19] T. Mitchell, "Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression," 2015. [Online]. Available: http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf.

[20] A. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems 14*, Massachuset Institute of Technology, 2002, pp. 841-848.

[21] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *The 2003 conference on Empirical methods in natural language processing*, Stroudsburg, 2013.

[22] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological Bulletin,* vol. 76, p. 378–382, 1971.

[23] A. Polyvyanyy, "Evaluation of a novel information retrieval model: eTVSM," in *Master's thesis, Hasso Plattner Institut*, 2007.

[24] O. Karnalim, "Extended Vector Space Model with Semantic Relatedness on Java Archive Search Engine," *Jurnal Teknik Informatika dan Sistem Informasi (JuTISI),* vol. 1, no. 2, pp. 111-122, 2015.

[25] O. Karnalim, "Improving Scalability of Java Archive Search Engine through Recursion Conversion and Multithreading," *CommIT (Communication and Information Technology) Journal,* vol. 10, no. 1, pp. 15-26, 2016.