Classification of Machine Learning Engines using Latent Semantic Indexing

Yuhanis Yusof[1], Taha Alhersh[2], Massudi Mahmuddin[3] and Aniza Mohamed Din[4]

[1]*Universiti Utara Malaysia, Malaysia, yuhanis@uum.edu.my*
[2]*Universiti Petronas Malaysia, Malaysia, taha.trh@gmail.com*
[3]*Universiti Utara Malaysia, Malaysia, ady@uum.edu.my*
[4]*Universiti Utara Malaysia, Malaysia, anizamd@uum.edu.my*

ABSTRACT

With the huge increase of software functionalities, sizes and application domain, the difficulty of categorizing and classifying software for information retrieval and maintenance purposes is on demand. This work includes the use of Latent Semantic Indexing (LSI) in classifying neural network and k-nearest neighborhood source code programs. Functional descriptors of each program are identified by extracting terms contained in the source code.Inaddition, information on where the terms are extracted from is also incorporated in the LSI. Based on the undertaken experiment, the LSI classifier is noted to generate a higher precision and recall compared to the C4.5 algorithm as provided in the Weka tool.

Keywords — Latent Semantic Indexing, Software Classification; C4.5,Machine Learning Algorithms

I. INTRODUCTION

Document classification has always been an important application for information retrieval. It can improve the speed of information retrieval and aid in locating and obtaining the desired information rapidly and accurately. Nowadays, due to the development of information technology, extensive studies have been conducted on document classification. Automatic software classification became one of the most important topics in software engineering area (Kawaguchi, Garg, Makoto, & Inoue, 2002). This is because of the new problems occurred upon constructionof software archives. For instance in 2002, the *SourceForge.net* had over seventy thousand registered software (Kawaguchi, Garg, Makoto, & Inoue, 2004). As this repository receives input (i.e. software files) from various developers whom have various backgrounds, categorizing the packages relies heavily on the textual provided and/or contained in them. One issue which arises from such situation is the involvements of human which may be subjective. Existing approaches that adopts manual classification require more time and high level of software understanding (Kawaguchi et al, 2002). This is because of the large size code embedded in software and the ambiguous code specification. Hence, the classification tasks are very time consuming. Additionally, inconsistent classification resultsmay occur due to more than one employee organizing the files.

This work tries to overcome such problem by introducing the use of Latent Semantic Indexing (LSI) that utilizes terms extracted from source code program for classification purposes. The LSI information retrieval model builds upon the prior research in information retrieval and, using the singular value decomposition (SVD)(Golub& Loan, 1996) to reduce the dimensions of the term-document space. Such an attempt is seen to solve the synonomy and polysemy problems that affect automatic information retrieval systems. In this work, the LSI relies on the constituent terms of the source code program to suggest the program's semantic content.

The undertaken work is a preliminary experiment to investigate the utilization of LSI on functional descriptors of source code programs in determining the domain of a program.Furthermore; it is to identify whether the LSI approach is better compared to existing work that employs decision tree, C4.5. This paper is structured as follows; in section II, we present brief information on existing work in software classification along with Latent Semantic Indexing. Section III includes description on how the work was performed and this is followed with a discussion on the obtained results in section IV. Finally, we conclude the work in section V that also contains the future work.

II. SOFTWARE CLASSIFICATION

There are many source code uploaded on the Internet that can be accessed through various web sites such as SourceForge, Plant Source Code and Free code(Korvetz, Ugurel, & Giles,

2003).Software classification plays a role in the field of software reusability(Poulin&Yglesias, 1993). For instance70% of software development budgets are spent on software maintenance, so the need of classifying the software to a particular type became an important topic to help in making accurate decision on code changes(Phillips & Black, 2005). Software classification helps to order software components in one repository into specific groups. With this, similar components can be grouped in the same category depending on the functionality of these components (Merkl, 1995).

Code metric histograms and genetic algorithms have been used to develop the Author Identification Software that identifies the original author(Lange & Mancoridis, 2007). 14 variables have been specified such as the way of typing the name of the functions and code specifications. Also software metrics were used to portray specified variables into histograms and later studied the histograms to identify the author(Lange & Mancoridis, 2007).

Recent work that also utilizes software metrics in source code classification is reported by (Yusof & Ramadan, 2010) and (Lerthathairat & Prompoon, 2011). In the former work, the researchers classify source code programs using classifiers included in WEKA. Three software metrics were used to automatically classify software packages, namely the Line of Codes (LOC), McCabe's Cyclomatic Complexity (MVG) and Weighted Methods per Class (WMC1). On the other hand, the work presented in (Lerthathairat & Prompoon, 2011) focuses on software metrics and fuzzy logic to improve code quality with refactoring techniques. They classify bad smell, clean code and ambiguous code.

To classify source code programs into categories, existing software classification approach also utilizes the Comments and specification, source code variables and Readme files(Korvetz, et al., 2003). Another work done in software classification is discussed in (Jianhui, 2008).They classify malicious samples into categories using three phases: Analyzing an object, Represent and store the knowledge and self learning from the new objects.

A. Latent Semantic Indexing

Latent Semantic Indexing reduces the vector space by creating a subspace of the matrix dimensions in order to remove noise and redundant terms. The reduced space presents a meaningful association between terms that in turn relate document(Kosala&Blockeel, 2000). The first step is to index frequently occurring terms in a term-document matrix and compute singular value decomposition (SVD) from the original k-dimensional term-document matrix. SVD is a matrix decomposition method commonly used for data analysis. The original term-document matrix, X, is decomposed into several matrices so their features can be revealed, for example document-document relationships. The decomposition is expressed as,

$$X(SVD) = T_{t\times k} \cdot S_{k\times k} \cdot D_{k\times d}$$

where, T is a left singular vector representing a term by dimension matrix, S is a singular value dimension by dimension matrix and D is a right singular vector representing document by document matrix(Kontostathis&Pottenger, 2003). The decomposed matrices are then truncated into a dimension less than the original k-value and the original X matrix approximated in the reduced latent space which better represents semantic relationships between terms compared to the original k-dimension document space.

A work that utilizes LSI in document classification can be seen in (Kosala & Blockeel, 2000). They extend the use of existing LSI by integrating information on the document ontologies. Such an approach is believed to improve knowledge extraction from web resident documents.

In the work done by (Cheng Hua & Soon Cheol, 2007), they construct document classification systems using artificial neural network that is integrated with LSI. The experimental evaluations show that the system training with the LSI is considerably faster than the original system training with the Vector Space Model and that the former yields better classification results. There are two differences between our work and theirs. First, we are using LSI independently and second we are utilizing LSI on a semi structured document. Hence, terms contained in the document may have different weighting.

A recent work on LSI in document classification is as reported by (Liping et al, 2010). They proposed a compact document representation with term semantic units which are identified from the implicit and explicit semantic information. The implicit semantic information is extracted from syntactic content via LSI while the explicit semantic information is mined from the external semantic resource namely the *Wikipedia*.

III. MATERIALS AND METHODS

This section provides information on how the work was undertaken. There are 4 steps involved; data collection, data preprocessing, development of LSI matrix and evaluation.

A. Data Collection

In the first stage, we downloaded 100 programs of neural network and k-nearest neighborhood, respectively, from software repositories (e.g. *SourceForge.net* and *Koders.com*).These programs are then stored in separate folders. From the obtained programs, we only include 90% of the programs while the remaining 10 programs from each category will later be used as the testing dataset.

B. Data Preprocessing

In order to extract the functional descriptors terms (terms contained in the source code program), we utilizes a code parser that is able to extract each term separately from each line in the program. This parser is able to operate on two programminglanguages which are the C and Java language. Prior to utilizing the extracted term, we performed two other processes. The first process is stemming which is done using Porter Stemmer algorithm (Porter, 1997). Stemming algorithm is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalization process that is usually done when setting up information retrieval systems.In the second process, we discard common adjectives (big, late, high), frilly terms (therefore, thus, however, albeit, etc.), terms that appear in every source code program and that appear in only one program.

Using the list of content terms and programs, we later generate a *term-document matrix*. This matrix represents a very large grid, with programs listed along the horizontal axis, and content terms along the vertical axis. For each term in the list, we go across the appropriate row and put '1' in the column for any program where that term appears. If the term does not appear, we assign '0'. We then obtain a numerical grid with a sparse scattering of 1.

In order to better represent the extracted terms, we also employ the local and global weighting. Termsthat frequently appear in a program and are at specific location (for example a term found as a class name is more important compared to the one found in a comment statement) are given a greater local weight than terms that appear once. We use a formula called *logarithmic local weighting* to generate our actual value. On the other hand, the global weighting applies to the set of all programs in our collection. Such a weighting indicates that terms that appear in only a few programs are likely to be more significant than terms that are distributed widely across the collection. In this work, we employ the *inverse document frequency* to calculate global weights.

C. LSI Matrix Development

Once the final term-document matrix is constructed, we need the Singular Value Decomposition(Golub& Loan, 1996)of this matrix in order to construct a semantic vector space that can be used to represent conceptual term-document associations. Such decomposition projects the large multidimensional space down into a smaller number of dimensions. In doing so, terms that are semantically similar will get squeezed together, and will no longer be completely distinct.

D. Evaluation

In order to evaluate the LSI classifier, we compare its performance against the classification made using decision tree C4.5. A total of 10 source code programs from both Neural Network and K-nearest neighborhood categories (which were not used in constructing the LSI matrix) are utilized as the testing dataset.

Precision and recall are the two measurementsused to evaluate the classification accuracy. Precision is the proportion of relevant instances in the results returned. For instance, if the precision is 0.72 then it means that 72% of returned instances were relevant. On the other hand, recall values represent the ratio of relevant instances found to the total of relevant instances(Pumpuang, et al, 2008).

IV. RESULTS

The obtained result is depicted in Tables 1 and 2. Table 1 contains the precision and recall for dataset involving neural network programs while Table 2 depicts the related values for k-nearest neighborhood source code programs.

Table 1: Precision and Recall for Neural Network Programs

	LSI		C4.5	
	Precision	Recall	Precision	Recall
Q1	0.60	0.80	0.50	0.75
Q2	0.60	0.80	0.50	0.74
Q3	0.80	0.85	0.70	0.59
Q4	0.80	0.85	0.60	0.79
Q5	0.70	0.68	0.80	0.72
Q6	0.80	0.79	0.70	0.75
Q7	0.90	0.83	0.80	0.80
Q8	0.50	0.83	0.50	0.81
Q9	0.70	0.77	0.70	0.71
Q10	0.70	0.78	0.60	0.78

Table 2: Precision and Recall for K-nearest Neighborhood Programs

	LSI		C4.5	
	Precision	Recall	Precision	Recall
Q1	0.90	0.92	0.80	0.90
Q2	0.80	0.79	0.80	0.75
Q3	0.80	0.85	0.80	0.86
Q4	0.80	0.85	0.80	0.83
Q5	0.70	0.71	0.70	0.70
Q6	0.80	0.84	0.80	0.83
Q7	0.90	0.93	0.80	0.90
Q8	0.60	0.75	0.70	0.74
Q9	0.80	0.84	0.80	0.82
Q10	0.80	0.84	0.80	0.85

In all of the testing programs, LSI has generated at least equal precision with C4.5 except for Q5 in Neural Network and Q8 in K-nearest neighborhood domain. We also illustrate the average of precision and recall values in Figure 1 and Figure 2 respectively. In both figures, it is noted that LSI generates a higher precision and recall values compared to C4.5.

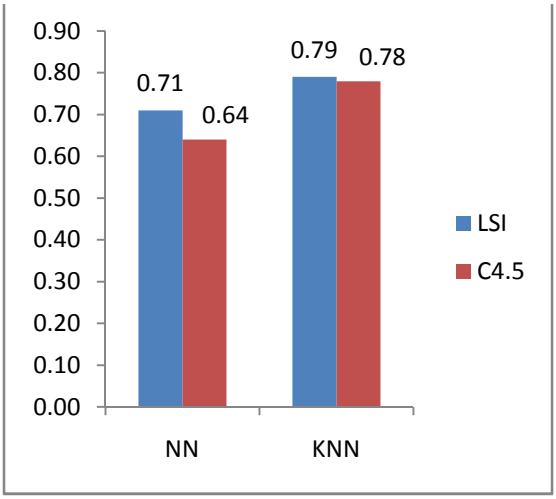

Figure 1: Precision - LSI vs. C4.5

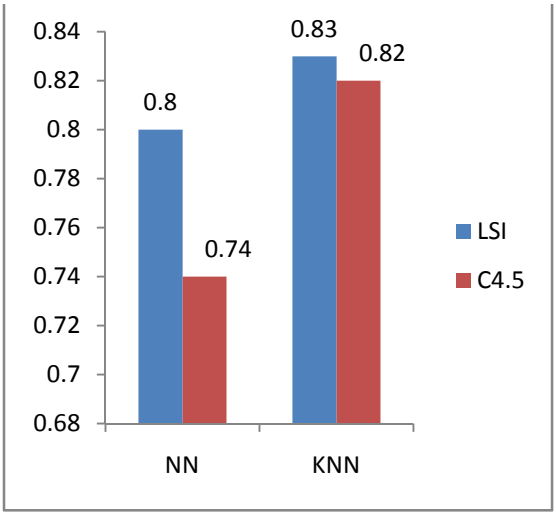

Figure 2: Recall - LSI vs. C4.5

V. CONCLUSION

In this work, we present the use of Latent Semantic Indexing that operates on terms extracted from source code programs. In addition, we utilize structure descriptors (that is the location of where the terms are extracted from) in calculating the local weight of the terms. It is learned from the undertaken experiments that LSI that integrates both functional and structural descriptors is a better classifier compared to a decision tree such as C4.5.

Further work needs to be done to improve the classification accuracy of LSI. This includes the use of other structure descriptors of source code programs. For example the data of software metric such as depth of inheritance tree and coupling be-

tween objects may be useful in differentiating between machine learning engines.

REFERENCES

Cheng Hua, L., & Soon Cheol, P. (2007, Nov).*Artificial Neural Network for Document Classification Using Latent Semantic Indexing*. Paper presented at the International Symposium on Information Technology Convergence, .

Golub, G. H., & Loan, C. F. V. (1996).*Matrix Computations* (3rd ed.): John Hopkins University Press.

Jianhui, L. (2008). *On malicious software classification*. Paper presented at the International Symposium on Intelligent Information Technology Application Workshops.

Kawaguchi, S., Garg, P. K., Makoto, M., & Inoue, K. (2002).Automatic categorization algorithm for evolvable software archive.*Six International Workshop on principles of Software Evolution*, 195-200. Retrieved from

Kawaguchi, S., Garg, P. K., Makoto, M., & Inoue, K. (2004). MUDABlue: An Automatic Categorization System for Open Source Repositories. *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, 184-193. Retrieved from

Kontostathis, A., & Pottenger, W. M. (2003).*A framework for understanding LSI performance*. Paper presented at the Proceedings of ACM SIGIR Workshop on Mathematical /Formal Methods in Information Retrieval.

Korvetz, R., Ugurel, S., & Giles, C. (2003).*Classification of Source Code Archive*. Paper presented at the 26th annual international ACM SIGIR conference on Research and development in information retrieval.

Kosala, R., & Blockeel, H. (2000). Web Mining Research: A Survey. *SIGKDD Explorations, 2*(1), 1-15.

Lange, R., & Mancoridis, S. (2007). *Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics*. Paper presented at the Proceedings of the 9th annual conference on Genetic and evolutionary computation.

Lerthathairat, P., & Prompoon, N. (2011, May).*An Approach for Source Code Classification to Enhance Maintainability*. Paper presented at the Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE).

Liping, J., Jiali, Y., Jian, Y., & Houkuan, H. (2010).*Text Clustering via Term Semantic Units*. Paper presented at the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT).

Merkl, D. (1995, Nov/Dec 1995). *Content-based software classification by self-organization.* Paper presented at the Proceedings of the IEEE International Conference on Neural Networks.

Phillips, N., & Black, S. (2005). *Distinguish between Learning, Growth and Evolution*. Paper presented at the IEEE International Workshop on Software Evolution.

Porter, M. F. (1997). An algorithm for suffix stripping. In J. P. Karen Sparck, Willett (Ed.), *Readings in Information Retrieval* (pp. 313-316): Morgan Kaufmann Publishers Inc.

Poulin, J. S., & Yglesias, K. P. (1993).*Experiences with a faceted classification scheme in a large reusable software library (RLS)*. Paper presented at the Seventeenth Annual International Computer Software and Application Conference.

Pumpuang, P., Srivihok, A., & Praneetpolgrang, P. (2008).*Comparisons of Classifier Algorithms: Bayesian Network, C4.5, Decision Forest and NBTree for Course Registration Planning Model of Undergraduate Students*. Paper presented at the Conference on Systems, Man and Cybernetics.

Yusof, Y., & Ramadan, Q. H. (2010).Automation of Software Artifacts Classification.*International Journal of Soft Computing, 5*(3), 109-115.