Research Journal of Applied Sciences 7 (5): 266-271, 2012
ISSN: 1815-932X
© Medwell Journals, 2012

Source Code Classification using Latent Semantic Indexing with Structural and Frequency Term Weighting

Yuhanis Yusof, Taha Alhersh, Massudi Mahmuddin and Aniza Mohamed Din
School of Computing, UUM College of Arts and Sciences, Universiti Utara Malaysia,
06010 UUM Sintok, Kedah, Malaysia

Abstract: In recent years, there is an increase in the number of open source software. Hence, the demand for automatic software classification is also increasing. Latent Semantic Indexing (LSI) is an information retrieval approach that is utilized in classifying source code programs. This research proposes a Latent Semantic Indexing classifier that integrates information on structural and frequency of terms in its weighting scheme. The content terms are identified by extracting words in the source code program. Based on the undertaken experiment, the LSI classifier is noted to generate a higher precision and recall compared to the C4.5 algorithm. Furthermore, it is also learned that the use of structural information in the weighting scheme contribute to a better classification.

Key words: Latent semantic indexing, software classification, C4.5, term weighting, algorithm, synonomy

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. Now a days due to the development of information technology, the demand for automatic classification is increased. Automatic software classification became one of the most important topics in software engineering area (Kawaguchi *et al*., 2003). This is because of the new problems occurred upon constructing of software archives. For instance in 2002, the SourceForge.net had over seventy thousand registered software (Kawaguchi *et al*., 2004). As this repository receives input (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*., 2003). This is because of the large size code embedded in software and the ambiguous code specification. Hence, the developers need to spend their time and efforts to know the specific category that software belongs to. The process of organizing the files may be carried out by a number of employees and may resulted on unreliable classification. This research tries to overcome such problem by introducing the use of Latent Semantic Indexing (LSI) (Deerwester *et al*., 1990) 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 and Van 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 research, the LSI relies on the constituent terms of the source code program to suggest the program's semantic content.

The undertaken research is an experiment to investigate the utilization of both functional and structural information in source code classification. Researchers would like to identify if a structural and frequency term weighting scheme is more beneficial than using term frequency on its own. Furthermore, the research is also undertaken to see if LSI is a better classifier as compared to a decision tree.

Software classification: Every day, there are many software source code uploaded on the internet so, the web now days contains different sets of source codes

Corresponding Author: Yuhanis Yusof, School of Computing, UUM College of Arts and Sciences, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia

which can be reached by using source code web sites such as SourceForge, Plant Source Code and Free Code (Korvetz *et al.*, 2003). Software classification plays a role in the field of software reusability (Poulin and Yglesias, 1993). For instance, 70% 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 and 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 and Mancoridis, 2007). About 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 and Mancoridis, 2007).

Recent research that also utilizes software metrics in source code classification is reported by Yusof and Ramadan (2010) and Lerthathairat and Prompoon (2011). In the former research, 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 research presented in Lerthathairat and 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 research 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.

Latent semantic indexing: Latent Semantic Indexing was proposed by Deerwester *et al.* (1990) and is a method that integrates vector space model and Singular Value Decomposition (SVD). LSI 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 and 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 = A left singular vector representing a term by dimension matrix

S = A singular value dimension by dimension matrix

D = A right singular vector representing document by document matrix (Kontostathis and Pottenger, 2003)

The decomposed matrices are then truncated to a dimension less than the original k-value and the orginal X matrix approximated in the reduced latent space which better represents semantic relationships between terms compared to the original k-dimension document space.

A research that utilizes LSI in document classification can be seen in (Kosala and Blockeel, 2000). They extend the use of existing LSI by integrating information on the document ontologies. Such an approach is believe to improve knowledge extraction from web resident documents. Similar, approach is taken in the work nevertheless, researchers are utilizing structural information of terms contained in the program source code. And this information is incorporated in a term weighting scheme.

In the research done by Li and Park (2007) they construct document classification systems using artificial neural network that is intergrated 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, researchers 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 research 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 externalsemantic resource (Wikipedia).

There is also an issue in Utilizing Vector Space Model that is the term weighting. This is to assign weight to term for a document so that the assigned weight reflects the contribution of the term in differentiating the document from other documents existing research such as Zaman and Brown (2010) utilizes available term weighting schemes such as the Raw Frequency, Tf-Idf and Log Entropy.

MATERIALS AND METHODS

The architecture of the LSI classifier is illustrated in Fig. 1. There are four processes involved: data collection, data preprocessing, matrix generation and Singular Value Decomposition (SVD) generation.

Data collection: In the first process, researchers gather 100 programs of neural network and k-nearest neighborhood, respectively from various repositories (SourceForge). These programs are then stored in separate folders. From the obtained programs, researchers only include 90% of the programs while the remaining 10 programs from each category will later be used as the testing dataset.

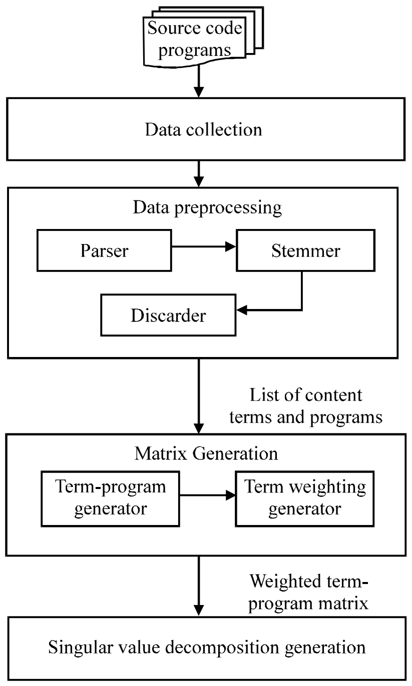

Fig. 1: Architecture of LSI with weighted term-program matrix

Data preprocessing: In order to extract the functional descriptors terms (terms contained in the source code program), researchers utilizes a code parser that is able to extract each term separately from each line in the program. Currently, the parser operates on 2 programming langauges which are the C and Java language. Prior to utilzing the extracted term, researchers performed 2 other sub-processes. First, we remove the commoner morphological and inflexional endings from words in English. This is known as stemming and is undertaken based on the Porter Stemmer algorithm (Porter, 1980). Upon completing the stemming process, we discard common adjectives (such as big, late and high), frilly terms (such as therefore, thus, however), terms that appear in every source code program and that appear in only one program file.

Using the list of content terms and programs, we later generate aterm-program 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. Terms that 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 calledlogarithmic local weighting to generate the actual value. On the other hand, the global weighting applies to the set of all programs in the 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 research, we employ the inverse document frequency to calculate global weights.

Singular value decomposition generation: Once the weighted term-program matrix is constructed, we need the Singular Value Decomposition (Golub and Van Loan, 1996) of this matrix in order to construct a semantic vector space that can be used to represent conceptual term-program 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.

Evaluation: In order to evaluate the LSI classifier, researchers compare its performance against a LSI classifier that only includes information on term frequency while calculating its weight. Later, a comparison against the classification made using decision tree C4.5 is also performed. 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 measurements used 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).

RESULTS AND DISCUSSION

In the first experiment, results of precision and recall for both categories of programs are shown in Table 1 and Table 2. Both of the tables contain results obtained using LSI with structural and frequency term weighting and LSI with frequency term weighting. It can be seen that the utilization of structural information (location of where the terms are extracted from) is beneficial as it produces higher precision and recall values compared to the one using only frequency information. The average precision value for KNN programs when both structural and frequency information is utilized is 0.79. This value is 0.01 higher to the one obtained using only frequency term weighting. On the other hand, in classifying NN programs, the proposed LSI classifier outperformed the precision generated by the frequency weighting LSI in 7 queries.

Upon obtaining results as shown in Table 1 and 2, researchers performed a comparison between LSI with structural and frequency term weighting and decision tree C4.5. Result for the said comparison is shown in Table 3 and 4. Table 4 shows the precision and recall for dataset involving neural network programs while Table 2 depicts the related values for k-nearest neighborhood source code programs. In all of the testing programs, LSI has generated at least equal precision with C4.5 except for

Table 1: Results between structural_frequency weighting and frequency weighting for neural network programs

	Structural_frequency weighting		Frequency weighting	
Programs	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: Results between structural_frequency weighting and frequency weighting for k-nearest neighborhood programs

	Structural_Frequency weighting		Frequency weighting	
Programs	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

Table 3: Precision and recall for neural network programs

	LSI		C4.5	
Programs	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 4: Precision and Recall for K-nearest Neighborhood Programs

	LSI		C4.5	
Programs	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

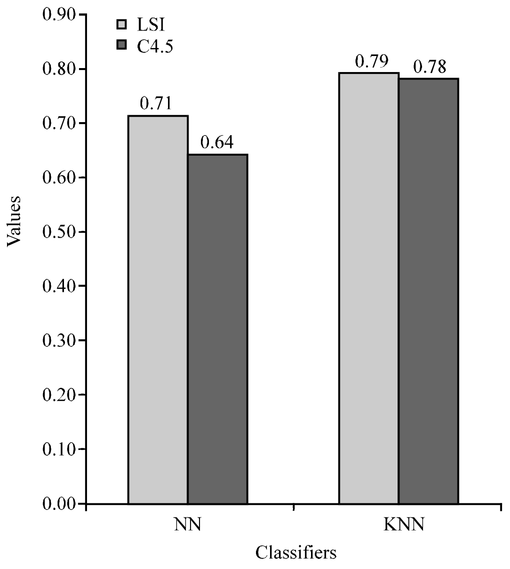

Fig. 2: Precision-LSI vs. C4.5

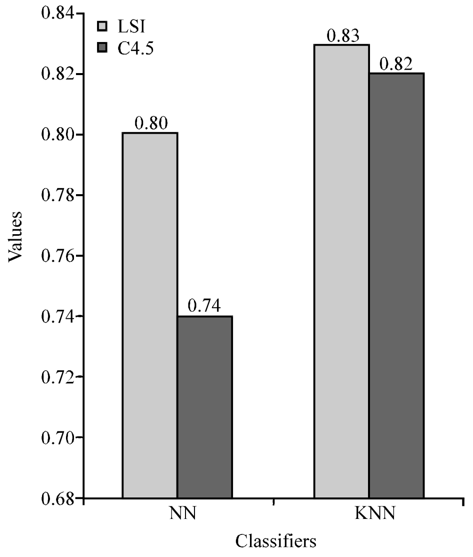

Fig. 3: Recall-LSI vs. C4.5

Q5 in Neural Network and Q8 in K-nearest neighborhood domain. We also illustrate the average of precision and recall values in Fig. 2 and 3, respectively. In both figures, it is noted that the proposed LSI generates a higher precision and recall values compared to C4.5.

CONCLUSION

In this research, researchers present the use of Latent Semantic Indexing that operates on terms extracted from source code programs. In addition, researchers utilize structural information contained in a source code program in calculating the term weighting. Such an approach is proven to help in producing a better classification. Furthermore, the use of LSI as a classifier is a better choice compared to a decision tree such as C4.5.

However, more research can be done to improve the classification accuracy of LSI. This includes the use of other structure descriptors of source code programs. This includes the relationships between objects or source code files for a particular application. In addition, focus can also be put in utilizing swarm computing approach in identifying term similarity.

REFERENCES

Deerwester, S., S.T. Umais, G.W. Furnas, T.K. Landauer and R. Harshman, 1990. Indexing by latent semantic analysis. J. Am. Soc. Inform. Sci. Technol., 41: 391-407.

Golub, G.H. and C.F. Van Loan, 1996. Matrix Computations. 3rd Edn., Johns Hopkins University Press, Baltimore.

Jianhui, L., 2008. On malicious software classification. Proceedings of the International Symposium on Intelligent Information Technology Application Workshops, December 21-22, 2008, IEEE Computer Society, Washington, DC, USA., pp: 368-371.

Kawaguchi, S., P.K. Garg, M. Matsushita and K. Inoue, 2003. Automatic categorization algorithm for evolvable software archive. Proceedings of the 6th International Workshop on Principles of Software Evolution, September 1-2, 2003, IEEE Computer Society Washington, DC, USA., pp: 195-200.

Kawaguchi, S., P.K. Garg, M. Matsushita and K. Inoue, 2004. MUDABlue: An automatic categorization system for open source repositories. Proceedings of the 11th Asia-Pacific Software Engineering Conference, November 30-December 3, 2004, Busan, Korea, pp: 184-193.

Kontostathis, A. and W.M. Pottenger, 2003. A framework for understanding LSI performance. Proceedings of the ACM SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval, July 28, 2003, Toronto, Canada.

Korvetz, R., S. Ugurel and C.L. Giles, 2003. Classification of source code archive. Proceedings of 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28-August 1, 2003, ACM, pp: 425-426.

Kosala, R. and H. Blockeel, 2000. Web mining research: A survey. ACM SIGKDD Explorat. Newslett., 2: 1-15.

Lange, R.C. and S. Mancoridis, 2007. Using code metric histograms and genetic algorithms to perform author identification for software forensics. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, July 7-11, 2007, ACM, pp: 2082-2089.

Lerthathairat, P., and N. Prompoon, 2011. An approach for source code classification to enhance maintainability. Proceedings of the 8th International Joint Conference on Computer Science and Software Engineering, May 11-13, 2011, Chicago.

Li, C.H. and S.C. Park, 2007. Artificial neural network for document classification using latent semantic indexing. Proceedings of the International Symposium on Information Technology Convergence, November 23-24, 2007, IEEE Computer Society, Washington, DC, USA., pp: 17-21.

Liping, J., Y. Jiali, Y. Jian and H. Houkuan, 2010. Text clustering via term semantic units. Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology, December 9-12, 2010, Sydney, NSW, Australia.

Merkl, D., 1995. Content-based software classification by self-organization. Proceedings of the IEEE International Conference on Neural Networks, November-December, 1995, Perth, WA, Australia, pp: 1086-1091.

Phillips, N. and S. Black, 2005. Distinguish between learning, growth and evolution. Proceedings of the IEEE International Workshop on Software Evolution, September 26, 2005, London, South Bank Univeersity, UK., pp: 49-52.

Porter, M.F., 1980. An Algorithm for Suffix Stripping. Morgan Kaufmann Multimedia Information and Systems Series. Morgan Kaufmann Publishers Inc., UK., pp: 313-316.

Poulin, J.S. and K.P. Yglesias, 1993. Experiences with a faceted classification scheme in a large reusable software library (RSL). Proceedings of the 17th Annual International Computer Software and Applications Conference, November 1-5, 1993, Phoenix, AZ, USA., pp: 90-99.

Pumpuang, P., A. Srivihok and P. Praneetpolgrang, 2008. Comparisons of classifier algorithms: Bayesian network, c4.5, decision forest and nbtree for course registration planning model of undergraduate students. Proceedings of the Conference on Man and Cybernetics Systems, October 12-15, 2008, Bangkok, pp: 3647-3651.

Yusof, Y. and Q.H. Ramadan, 2010. Automation of software artifacts classification. Int. J. Soft Comput., 5: 109-115.

Zaman, A.N.K. and C.G. Brown, 2010. Latent semantic indexing and large dataset: Study of term-weighting schemes. Proceedings of the 5th International Conference on Digital Information Management, July 5-8, 2010, Ontario, Canada.