Association for Information Systems
AIS Electronic Library (AISeL)

SAIS 2016 Proceedings | Southern (SAIS)

2016

Automatic Clustering of Source Code Using Self-Organizing Maps

William Wilson
Kennesaw State University, wwilso48@students.kennesaw.edu

Jean-Jacques Muteteke
Kennesaw State University, jjmuteteke@yahoo.com

Lei Li
Kennesaw State University, lli13@kennesaw.edu

Follow this and additional works at: http://aisel.aisnet.org/sais2016

Recommended Citation
Wilson, William; Muteteke, Jean-Jacques; and Li, Lei, "Automatic Clustering of Source Code Using Self-Organizing Maps" (2016). *SAIS 2016 Proceedings*. 10.
http://aisel.aisnet.org/sais2016/10

This material is brought to you by the Southern (SAIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in SAIS 2016 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

AUTOMATIC CLUSTERING OF SOURCE CODE USING SELF-ORGANIZING MAPS

William Wilson
Kennesaw State University
wwilso48@students.kennesaw.edu

Jean-Jacques Muteteke
Kennesaw State University
jjmuteteke@yahoo.com

Lei Li
Kennesaw State University
lli13@kennesaw.edu

ABSTRACT

Source code classification is an important step in archiving and reusing the code. Given the complex nature of software, source code is often organized into categories manually by field experts. Such categorization process not only requires a pre-existing category schema, but also is labor intensive which is difficult to keep up with the fast-growing available source codes. In this paper, we proposed an innovative method that can automatically classify a set of source codes into clusters based on similarity of their functionalities. We used a neural-network-based algorithm, Self-Organizing Maps (SOM), to cluster a list of source code extracted from an open-source software application site, SourceForge (sourceforge.net). Experiments have been conducted to test the feasibility of our approach. The research results showed SOM can automatically and effectively cluster source code with proper training. The implication of this study is discussed.

Keywords

Source code classification, automatic clustering, code reuse, self-organizing maps

INTRODUCTION

Code reuse is a useful tool for reducing the cost of software projects and for teaching software development (Stiller and Leblanc 2006). Despite the possible benefits of code reuse, it has had limited practical success for a variety of reasons (Frakes and Kang 2005). One of those reasons is the difficulty of finding reusable code fits to the task. To facilitate the search of code, it is important that we can classify available source codes quickly and accurately. Although classification can be done by the field experts, this process is often time consuming. As the number of source codes increases every day, human-based classification or clustering has become more and more costly and even impractical.

In this paper, we proposed to use a neural network based clustering algorithm, Self-Organizing Maps (SOM) (Kohonen 1995), to classify software source codes with little human intervention.

LITERATURE REVIEW

Searching for source codes to reuse is an important part of a software development project. A 1997 case study by Singer et al. (1997) noted that "search is done far more often [by software engineers] than any other activity." A more recent study by Sadowski et al. (2015) found that Google developers wrote an average of 12 search queries to their internal code repository every weekday. To improve the accuracy of searches, the internal code repository needs to be well organized, e.g. classified.

Document classification is a well-established field, with a large number of techniques available for different types of categorization. Source code and text documents are similar enough that document classification schemes could be adapted for code classification. Ugurel et al. (2002) conducted a proof of concept experiment testing the accuracy of a set of machine learning models used for document classification, and applying the best model to code. They found that Support Vector Machines (SVMs) were able to classify code by category and language. SVMs are a machine learning model that uses supervised learning: training data is labeled with a desired category, and the system's accuracy is then tested using unlabeled testing data.

Other approaches to code classification focus on unique attributes of code. McMillan et al. (2011) tackled the problem of classification where source code is unavailable by extracting API references from executable files. Using SVM to classify a set of code, they found that using known terms gave the best result, but concluded that classification by API packages "are a good alternative to terms in the case when the terms are not available."

Classification often relies on existing category schemes while such schema may not be available. The objective of this project is to organize the source codes into groups based on their functionalities without prior knowledge of categories. We call such process as clustering. In order to cluster the source codes without much human intervention, unsupervised machine learning techniques are usually preferred.

Self-Organizing Maps (SOM) is a neural-network based algorithm that focusing on unsupervised machine learning. SOM algorithm can condense high-dimension data into a two dimensional map that represents the data set. SOM was widely used to cluster images, texts, documents, and other high dimensional data. For example, Li (2007) successfully grouped semi-structured web pages into clusters using SOM without human intervention.

In this research, we proposed to apply SOM algorithm in the domain of software source code and conducted experiment to test the feasibility of our approach.

RESEARCH METHOD AND RESULTS

Software source codes contains a list of documents: libraries, code files, readme files, and documentations etc. In essences, the source codes are semi-structured or unstructured textual and high-dimensional data. We argue that SOM can effectively categorize those source codes. The proposed research approach had four phases: data source selection, data pre-processing, clustering, and validation.

Data source selection. Our data samples were obtained from SourceForge (SoureForge 2015), an open source software application site. SourceForge was chosen because it offered a large sample pool and the source codes are categorized by application type and programming language. For simplicity, ten applications were chosen from five categories, all coded in C++, for a total of fifty. The pre-existing category schema can be used as correct answers to assess the effectiveness of machine clustering. At the time of experiment, we assumed that extracted source codes were un-clustered. The fifty software applications were randomly sorted into a training dataset (thirty-three applications) and a testing dataset (seventeen applications).

Data pre-processing. The objective of data pre-processing is to transform the raw data into a smaller dataset in a format that is acceptable to SOM while still can represent the original dataset. Each extracted application source codes contains large amount of information including readme file and tens of individual C++ files. It's a complex and difficult process to decide what portion of the raw data should be used for clustering. In fact, the data pre-processing task itself makes a standalone research project. Our study focused on proof of concept on clustering and we simplified the data pre-processing process.

For each application, we examined all files associated with it and manually selected a few ones that better represent the application. Then a list of keywords was extracted from the selected files. After that, common words were removed from the word list. The common word was composed of a generic list of commonly used words and words occurred in every data sample. Words that appeared in every data set were also removed. The word list was very long, we kept the top 50 words to represent the corresponding software application. As the result, each software application had a text file containing a list of keywords. At the end, each text file was tokenized in a format required by SOM.

Clustering. In this study, we used SOM as clustering engine. The SOM implementation we used has four parameters: x dimension, y dimension, neighborhood size, and final iteration. The performance of on the parameter values chosen for a certain application domain (Polani and Uthmann 1993). We created a SOM parameter pool by trying all combinations of values in a pre-defined range. The value range as shown in table 1, is adapted from Li 2007.

x dimension	y dimension	Neighborhood size	Final iteration
3 – 10	3 – 10	2 – 6	5000 – 60000

Table 1. SOM Parameter Value Range. (Note: the final iteration is increased by 5000).

The SOM clustering engine needs to be trained before it can be used for classification. The training process is illustrated in figure 1. The original data set was randomly divided into a training dataset (two third) and a testing dataset (one third). For the training dataset, a set of parameter values first was retrieved from the value pool and used by SOM to cluster the training dataset. An F-measure (explained in the next section) was calculated for the clustering results. The process repeated for the next set of parameter values in the pool. The output of the training is the parameter value set that has the best F-measure value. The trained SOM clustering engine then was used to cluster the testing dataset and the effectiveness of the result was evaluated.

Evaluation. The effectiveness of our proposed clustering approach can be measured by comparing the clustering results of SOM with the ones of field experts. The pre-existing category scheme from SourceForge was used the clustering results from domain experts. F-measure (Larsen and Aone 1999) (Stein and Eissen 2002), a standard evaluation metric in the field of information retrieval, was used as main performance metric. F-measure provided an overall estimate of the combined effect of

cluster recall and precision. F-measure is a value between 0 and 1. In general, the higher the f-measure value, the better the clustering result.

$$F-Measure=\frac{(BETA\text{^}2+1)*CR*CP}{(BETA\text{^}2*CP)+CR}$$

Note: CR-cluster recall. CP-cluster precision. BETA is the relative importance of Recall vs. Precision. We assume CR and CP are equally important and thus set BETA as 1.

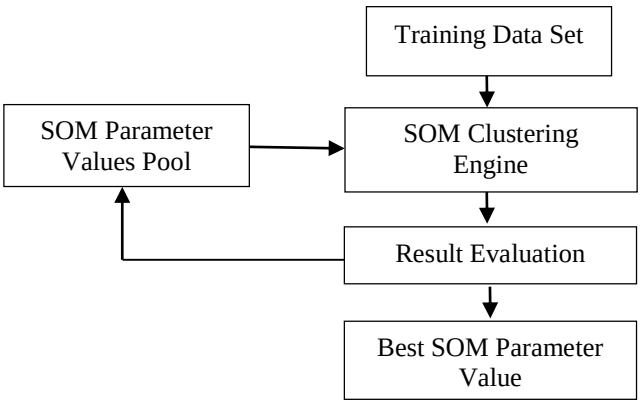

Figure 1. Program Flow of SOM Clustering Training Process

In our experiment, we were able to identify the best SOM parameter values for the training set with F-measure value of 0.36. Then we used those parameter values to cluster the testing data set and F-measure value is 0.32 for the resulting clusters.

CONCLUSION AND DISCUSSION

In this paper, we proposed a novel approach to automatically cluster software source codes using Self-Organizing Maps. We designed and implemented a proof of concept system. Our experiment results showed that, with proper training, our approach can effectively cluster a group of software application source codes extracted from SourceForge. Our approach contributes to the software development community by enabling more accurate search through better organization of the source codes. Oour approach also explored the applicability of SOM in a different domain.

Our study has several limitations. First of all, we made some assumptions and simplified data pre-processing phase since our focus was on the clustering. We manually reduce the large data set into small ones and the keywords extraction and selection was also streamlined with arbitrary decisions by the authors. As a result, the performance of the clustering engine is only moderately well with f-measure value of 0.32 for the selected testing data set. More research need to be done on selecting a list of keywords that can better represent the corresponding source codes in the clustering process. The efficiency of the data processing is also a concern especially when dealing with source codes containing lot of data. We are investigating ways to streamline the process and reduce the processing time.

Another limitation is the training of SOM. The performance of SOM relies on the parameter values it uses. In this study, we created a permutation of parameter values sets for SOM based on a pre-defined range for each parameter value. We need a more systematic approach to identify the best SOM parameter values for a given data set. We are investigating using genetic algorithm as the mechanism to fine the optimal or near optional SOM parameter values from a very large pool of value set candidates.

The study can be extended in the direction of the visualization of the clusters. The clustering result is presented in textual file which is fine for search engine to use, but it's not friendly for humans to view the categories and interact with. We are examining available tools to present clusters on a two dimensional map and allow users to interact with the clusters on the map.

REFERENCES

1. Frakes, W.B. and Kang K., (2005). "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, 31(7), July, pp. 529-536
2. Kohonen, T. (1995). Self-Organizing Maps. Berlin, Springer-Verlag.
3. Larsen, B. and A. Aone, 1999. Fast and Effective Text Mining Using Linear-time Document Clustering. Proc. of the Fifth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining.
4. McMillan,C. , Linares-Vasquez, M., Poshyvanyk, D., Grechanik, M. (2011). Categorizing Software Applications for Maintenance. Proceedings from *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 343-352. DOI:10.1109/ICMS.2011.6080801
5. Polani, D. and T. Uthmann, 1993. Training Kohonen Feature Maps in different Topologies: an Analysis using Genetic Algorithms. Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA.
6. Sadowski, C., Stolee, K. T., and Elbaum, S. (2015). How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (ESEC/FSE 2015). ACM, New York, NY, USA, 191-201. DOI=http://dx.doi.org/10.1145/2786805.2786855
7. Singer, J., Lethbridge, T., Vinson, N., and Anquetil, N. (1997). An examination of software engineering work practices. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research* (CASCON '97), J. Howard Johnson (Ed.).
8. Sourceforge (2015), http://sourceforge.net/
9. Stein, B. and S. M. Z. Eissen, 2002. Document Categorization with Major CLUST. 12th Annual Workshop On Information Technologies And Systems (WITS'02), Barcelona, Spain.
10. Stiller E. and LeBlanc C. (2006). Teaching software development by example. *J. Comput. Sci. Coll.* 21, 6 (June 2006), 228-237.
11. Ugurel, S., Krivetz, R; Giles, C. L. (2002). What's the Code?: Automatic Classification of Source Code Archives. Proceedings from *Knowledge Discovery and Data Mining, 2002 8th ACM SIGKDD International Conference on,* 632-638. ISBN: 1-58113-567-X/DOI: 10.1145/775047.775141