Using Latent Semantic Analysis for Automated Grading Programming Assignments

Kartinah Zen, D.N.F Awang Iskandar, Ongkir Linang
Faculty of Computer Science and Information Technology
Universiti Malaysia Sarawak 94300 Kota Samarahan, Sarawak, Malaysia
email: kartinah@fit.unimas.my, dnfaiz@fit.unimas.my, mrongkir@hotmail.com,

Abstract - **Traditionally, computer programming assignments are graded manually by educators. As this task is tedious, time-consuming and prone to bias, the need for automated grading tool is necessary to reduce the educators' burden and avoid inconsistency and favoritism. Recent researches have claimed that Latent Semantic Analysis (LSA) has the ability to represent human cognitive knowledge to assess essays, retrieving information, classification of documents and indexing. In this paper, we adapt LSA technique to grade computer programming assignments and observe how far it can be applied as an alternative approach to traditional grading methods by human. The grades of the assignments are generated from the cosine similarity that shows how close students' assignments to the model answers in the latent semantic vector space. The results show that LSA is not able to detect orders of computer programming and symbols; however, LSA is able to grade assignments faster and consistently, which avoid bias and reduces the time spent by human.**

Keywords –latent semantic analysis; computer programming assignments

I. INTRODUCTION

Grading programming assignments manually involves a lot of work and time consuming, where each program must be tested and the source code must be read thoroughly by the educators before returning them to the students. They have to examine each line of code to give adequate grades even if the program failed to execute (Cheang et al., 2003). Different human graders have the probability to assign different grades even though to the same assignment. This can happen due to fatigue, favoritism and inconsistency.

To minimize such problems in grading assignments, there is a need to have an automatic tool which is able to automatically grade the programming assignments. Existing tools for automated programming assignments grading provide various features such as grading of assignments and submissions, ranging from script-based execution of test cases to metrics-based evaluation. Examples of automatic grading tools are GAME (Blumenstein et al., 2004), Course Master (Foxley et al., 2001), ASSYST (Jackson & Usher, 1997), GradeBot and CAP (Schorsch, 1995). Although the grading tools are designed with sophisticated functions and features, there are some disadvantages in the tools. For example, GAME focuses very much on commenting and indentation which do not affect the functionality of a program. On the other hand, Course Master can grade multiple programming languages but has limited feedback to the students. Meanwhile, ASSYST can only grade some parts of the assignments automatically, in which most of the grading processes are still done by human. In case of CAP, the tool is purposely designed for novice programmer and only concentrated on three criterions that are syntax errors, logical errors and programming style errors. For example, a programming style is considered incorrect if some errors are found in commenting declarations, commenting formal parameters or indenting of nested statements.

All of the mentioned grading tools are very rigid and highly dependence to the model answers, such as specific structures, programming styles, parameters and commenting, which are not practically accurate in grading computer programming assignments. A more effective and flexible grading tool is required to overcome such restrictions. In this paper, an automatic method which adopted Latent Semantic Analysis (LSA) to grade programming assignment is proposed. The aim of this paper is to evaluate the performance and consistency of LSA to mark programming assignments with selected structures and compare the marks that are awarded by both LSA and human graders (educators).

LSA uses a term-document matrix to represent the occurrence of terms in documents. Each row of the matrix represents a term occurs across documents in a collection, while each column corresponds to each document in the collection. The terms are subjected to weighting (normalization) that reflects their importance where the terms that rarely appear are given higher weight. The matrix is then decomposed into another three matrices and their dimensions are reduced to remove noises in the documents.