Mining Student CVS Repositories for Performance Indicators

Keir Mierle
Dept. Electrical & Computer Engineering
University of Toronto
keir@cs.utoronto.ca

Kevin Laven, Sam Roweis, Greg Wilson
Dept. Computer Science
University of Toronto
{klaven,roweis,gvwilson}@cs.utoronto.ca

ABSTRACT

Over 200 CVS repositories representing the assignments of students in a second year undergraduate computer science course have been assembled. This unique data set represents many individuals working separately on identical projects, presenting the opportunity to evaluate the effects of the work habits captured by CVS on performance. This paper outlines our experiences mining and analyzing these repositories. We extracted various quantitative measures of student behaviour and code quality, and attempted to correlate these features with grades. Despite examining 166 features, we find that grade performance cannot be accurately predicted; certainly no predictors stronger than simple lines-of-code were found.

1. INTRODUCTION

Version control repositories contain a wealth of detailed information about the evolution of a codebase. In this paper, we outline our experiences analyzing data from a large collection of CVS repositories created by many students working on a small set of assignments in a second year undergraduate computer science course at the University of Toronto.

We believe our data set is rather unique. It contains hundreds of completely independent repositories, one for each student. Each student is implementing the same thing at the same time. Previous work analyzing logs from version control systems has tended to focus on a single large repository involving many coders working on different parts of the same software project[5, 4].

1.1 Goals

The broad goal of our research programme was to extract information about student behaviour and code from version control repositories, in order to find statistical patterns or predictors of performance. It was our hope that these results can be used to identify and assist undergraduate students having difficulties. This paper outlines our attempts.

We attempted to identify work habits captured in the CVS repository that are indicative of strong or poor performance. We investigated such hypotheses as *students who start assignments early tend to do well*, and *students who submit assignments close to (or after) the deadline tend to do poorly*. Quantitatively confirming the effectiveness of good work habits could help encourage students to follow them.

In addition, we attempted to identify features of the code itself that are indicative of performance. As with work habits, confirming their effects on performance could be used to encourage students to write good code.

Finally, we were interested in finding early indicators of students who may be struggling in order to provide timely assitance. We hope to acheive this by finding a way to predict low grades (final course grades in the bottom third of the class) based on statistics extracted from a student's CVS repository.

1.2 CVS Background

CVS, the Concurrent Versions System[1], is a source code management system. It provides a facility for storing past and present versions of a project's codebase, as well as automating many aspects of writing software as a team.

A CVS repository consists of two parts: administrative files stored in a central location (known as `CVSROOT`), and RCS files associated with each file stored in the repository. The RCS files store revision histories for the individual files comprising the project.

This storage scheme is a bit complex to parse. To further complicate matters, in CVS there are two separate and occasionally disjoint records of activity in a repository: the `CVSROOT/history` file and the individual RCS files (ending in `,v`). File histories are implicitly stored in the `RCS` files which record modifications, additions, and scheduled deletions. The `history` file tracks (almost) all interactions between a user and a repository, including checkouts, updates, and conflict resolutions, as well as those listed above. Unfortunately, the history file does not record the initial importing (addition to the repository) of a project.

2. DATA PREPARATION

A combination of Python, ViewCVS[3], and MySQL helped massage the data into a more usable form. Our original code for extracting data from CVS repositories is based on ViewCVS, which includes a fast RCS parser written in C++. This code, including our modifications to the ViewCVS parser, is freely available at `www.cs.utoronto.ca/~keir/slurp-1.0.0.tar.gz`. The code parses every file in a repository and loads the transactions into a MySQL database for convienient access.

2.1 Transaction Clumping

One problem with CVS is the grouping of transactions. CVS does not keep any record of which operations were executed as part of a single client command. In order to reconstruct the use of the CVS repository, these must be re-grouped based on temporal proximity and other details of the transactions.

We used a variant of the sliding window approach[6] to clump groups of transactions into single events. In this approach, any set

of transactions with the same user and comment string, in which neighbouring transactions occur within τ seconds of each other are grouped into a single event. For our data, we found that $\tau = 50$ worked well. While this disagrees with the results in [6], this is not unexpected, as our repositories are very small, with most students working on the local network, leading to much shorter operation times.

Two modifications were made to the sliding window approach to improve results. First, it was noted that some CVS clients allow the user to enter a different comment for each file involved in a single event. To account for this, all transactions by the same user that occur at the same second were grouped together. Second, it was noted that a single file cannot have multiple modifications within one event. After the clumping was complete, any event which contained the same file more than once was split into separate events, decreasing the amount of over-clumping.

2.2 Feature Extraction

The quantitative and visual analysis techniques we intended to use require numerical data. Accordingly, we converted the known data about each student into a set of numerical summary statistics (called *features*), to be evaluated as predictors of student performance. Our system extracts 166 unique features from the transaction histories, log comments, and details of the actual code files from each student. In order to be able to quantitatively compare the effects of different features, each feature was normalized to have zero mean and unit variance across the entire dataset.

Three classes of features were extracted. The first features were calculated from the database of CVS data. These largely represent student behaviour and work habits. They include things like the average number of revisions per file, number of local CVS operations, number of update transactions, how close to the deadline they submit the assignment, and more.

The second group of features came from parsing the Python and Java code. For these features, each student's repository was checked out and examined. Two simple parsers were written (in Python) to extract features directly from the Python and Java code, such as the number of while loops, number of comments, and number of instances of certain formatting habits.

For the final group of features, we used PMD[2] to examine all Java files. PMD detects many types of higher-level features, particularly style violations or bad practices. It detects things like variable names that don't follow a naming standard, boolean expressions that can be simplified, empty if statements, asserts without a message, and a whole slew of others.

In addition to extracting these features, records of student academic performance were added to the database. Along with grades for each student, each student was labelled as being in the top, middle, or bottom third of the class.

3. VISUAL & QUANTITATIVE ANALYSIS

A variety of visual and quantitative analysis techniques were applied to the data. Visualizations used include views which aggregate statistics across all students and assignments, as well as specialized views which allow us to examine the behaviour of a single student and/or a single assignment. Quantitative analysis techniques used include examining the mutual information between each feature and the student grade, as well as the application of statistical pattern recognition algorithms for predicting grades from the features.

The following sections present the results of our analysis in terms of the three goals of the project: investigating the effects of work habit on grades, the effects of code quality on grades, and attempting to predict performance based on all of the information available.

3.1 Mutual Information for Feature Evaluation

The mutual information between two (discrete) random variables x and y is defined as the cross-entropy between their joint distribution and the product of their marginal distributions:

$$I(x,y) = KL[p(x,y)\|p(x)p(y)] = \sum_{x,y} p(x,y)\log_2 \frac{p(x,y)}{p(x)p(y)}$$

The mutual information between two random variables is a measure of their dependence or independence. It is a more stringent measure than the traditional correlation coefficient, which only measures average second order statistics but cannot capture complex higher order dependencies.

For our data, we did not have enough samples to accurately estimate the joint density between all the features we investigated. However, we did have enough data to estimate the mutual information between a single feature and student grades. Specifically, we created a binary random variable y which was 1 if a student achieved a grade which placed them in the top third of the class and zero if it placed them in the bottom third of the class. Students in the middle third were excluded from the estimation procedure. We then discretized each feature f into $K = 20$ bins (with equal sized ranges f_k) and computed the mutual information between this discretized random variable and the binary grade variable as follows:

$$I(f) = \sum_{y=0,1}\sum_{k=1}^{K} p(y)p(f \in f_k|y)\log_2 \frac{p(f \in f_k|y)}{p(f \in f_k)}$$

where $p(f \in f_k)$ is the overall observed frequency with which the feature falls into bin k and $p(f \in f_k|y)$ is the frequency for either the high or the low grade students. $p(y=1) = p(y=0) = 0.5$ since we select exactly equal numbers of students with high grades (the top third of the class) and low grades (the bottom third of the class).

Using these estimates, we can rank the features by how much information they contain about the course grade. figure 1 gives a short list of the top features and their estimated mutual information. The result was quite surprising to us:

Of the 166 features we examined, only 3 had significant correlation with grade. Of these, the most significant was the total number of lines of code written.

Given the number of student repositories used, only three features had statistically significant mutual information with the grade of the student at the $p = 0.05$ level: lines of code written by the student, number of commas followed by spaces[1] and total length of diff text[2] (In this case $I = 0.22$ bits was the significance cutoff.) These features are above the line in figure 1.

3.2 Effects of Work Habits on Grades

The first hypothesis investigated was that students who do well on the assignments will tend to do well on the final exam. Figure 2 shows a plot of term grade (from the assignments) versus exam grade, which suggests that such a relationship is present.

We compared student grades with the number of transactions of various types executed by the student, hoping to see that a certain

[1]Consider `foo(a,b,c,d)` instead of `foo(a, b, c, d)`.

[2]Each time a student does `cvs commit`, only changes to the code are stored. The "total diff length" feature is the total character count of all the deltas combined.

M.I. (bits)	Feature Description
0.29	newline characters in students files
0.28	times a space followed a comma, e.g. "`foo(a, b, c)`"
0.26	characters in diff text between successive revisions (CVS)
0.20	comments (Python)
0.20	literal strings (Python)
0.19	operators (Python)
0.16	characters in all comments
0.16	function definitions (Python)
0.14	`while` loops (Python)
0.14	Terminal tokens (Python)
0.13	4-space indents
0.13	comment-space-capital sequences e.g. "`// Formatted`"
0.12	commits (CVS)
0.12	`for` loops (Python)
0.11	newlines (Python)
0.11	files in repository
0.11	violations of "Assertions should include message" (PMD)
0.11	`self` references (Python)
0.10	modifies (CVS)
0.10	violations of "Avoid duplicate literals" (PMD)
0.10	`except` tokens (Python)
0.10	leading tabs
0.10	total transactions (CVS)
0.09	Average revisions per file (CVS)

Figure 1: Mutual information between various features of a student's repository and the binary indicator of whether they fall into the top third or bottom third of the class (by final grade). Each feature is a *count* of how many times it occurred, for example 'comments (Python)' is the number of comments a student had in their code. Features marked (Python) were drawn from the students' Python code. Likewise, the (CVS) marking means the feature was drawn from the CVS logs, and (PMD) denotes rule violations PMD found. Only values greater than 0.22 bits are statistically significant at the $p = .05$ level, given the number of students. Significant features are located above the horizontal line.

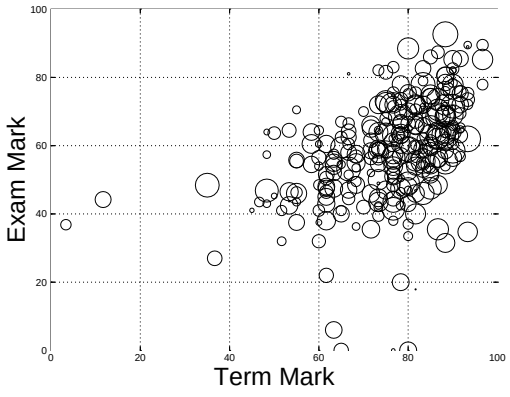

Figure 2: The relationship between the two components making up a final grade. *term mark* is the net mark on the coding assignments. Radius of each circle indicates the total number of CVS transactions executed by the student.

type of transaction (or mix of types) is particularly indicative of performance. The displays, however, suggest that no particular transaction types are indicative of high or low performance (see figure 6 below). This was confirmed by our mutual information analysis which shows that none of transaction type counts have statistically significant mutual information with grade.

Each of the features extracted from the work habits of the student was examined for a relationship with final course grade. While we anticipated several of these would have an impact, it turned out that none had mutual information with final grade that was statistically significant at the $p = 0.05$ level. In particular, we were surprised to note that how early a student starts assignments, and how close to the deadlines they submit, had essentially no predictive value for student grade. (Although some students may have started work on their home machines early but not checked into the CVS repository until the last moment.)

In fact, the only feature drawn from the CVS repository that had statistically significant mutual information with final grade was the total number of characters in the diff text between successive revisions. This, however, is actually an estimate of how much total code the student has written, as opposed to a feature of their work habits.

These results suggest that, contrary to the beliefs of many instructors, student work habits have very little effect on their performance, so long as they eventually do the work. Students who wait until the last minute to do an assignment appear just as likely to do well (or poorly) as those who both start and complete the assignment well ahead of time.

3.3 Effects of Code Quality on Grades

Both visual and quantitative analysis techniques were applied in an attempt to correlate the various features describing code quality with grade.

The feature with the strongest predictive value turned out to be lines of code written, as measured by the number of linefeed charcters in the files in each student's repository. Plots of grade versus lines of code are shown in figure 3. They show informally that students who write very little code tend to do poorly (but beyond a certain point writing more code does not correlate with higher grades) and that (with a few exceptions) students who do well on assignments also do well on the exam. We have also performed a more quantitative mutual information analysis (see below) showing that the number of lines of code written is a statistically significant (though weak) predictor of grade at the $p = 0.05$ level, and is a stronger predictor than any other complex feature we were able to find. Figure 7 provides an alternate view of this relationship, showing histograms of the lines of code written by students in the top third and bottom third of the class, as well as those written by the entire class.

This relationship is not surprising considering that students who do not write enough code to complete an assignment necessarily get low grades. Once a student writes enough code to finish an assignment, lines of code are no longer a strong indicator of quality.

One of the other two features that had statistically significant mutual information with the grade of students at the $p = 0.05$ level was also a code quality measure: the number of times a comma was followed by a space. This indicates care being taken in formatting code, and may well be an indication of the total time spend on the assignment by the student. It is true that some programmers prefer the `foo`(a,b) form; thus, if the code feature indicates the no-space form, no conclusion can be drawn from code formatting habits.

The fact that all three of the statistically significant indicators of performance were indicators of time spent on the assignments

suggests a simple conclusion:

In order to succeed in a course, students should invest the necessary time to complete assignments with care. It doesn't matter when they put this time in, so long as they do so.

3.4 A Machine Learning Approach to Grade Prediction

With numeric features in hand, we were ready to try a variety of statistical pattern recognition algorithms for predicting grades based on the features. Specifically, the algorithms were trained to distinguish between students in the top and bottom third of the class, with those in the middle third left out. Of course, the lack of significant mutual information between grades and most of the features we extracted did not bode well for such an enterprise, but we conducted several experiments nonetheless and report their results here.

We used three very basic algorithms from the machine learning and applied statistics fields: nearest neighbour classification, Naive Bayes, and logistic regression. Before we could classify, we normalized the features to have zero mean and unit variance. (We applied this normalization to all features, even those whose histograms were obviously not Gaussian.)

As expected, none of the classifiers were able to reliably predict grades for students based on the features given. While some algorithms managed to overfit the training sets and achieve 0% training error, the errors on an independently held out test set were always far inferior. A leave-one-out (LOO) cross validation estimate of test error was typically around 25%. In particular, for logistic regression, our LOO error was 29.7%, for Naive Bayes it was 23.9% (using discretized versions of the features), and for nearest neighbour it was also 23.9% (at $K = 21$ using Euclidean distance in the normalized feature space). In all these experiments, we used 166 normalized features to classify 69 students from the top third of the class (by grade) from 69 students from the bottom third.

4. CONCLUSION

We have described the results of analyzing data from a large collection of CVS repositories created by many coders, in this case students, working on a small set of identical projects (course assignments). We have implemented a complete system for parsing such repositories into a SQL database and for extracting, from the database and repositories, various statistical measures of the code and version histories.

Although version control repositories contain a wealth of detailed information both in the transaction histories and in the actual files modified by the users, we were unable to find any measurements in the hundreds we examined which accurately predicted student performance as measured by final course grades; certainly no predictor stronger than simple lines-of-code-written was found.

These results directly challenge the conventional wisdom that a repository contains easily extractable predictive information about external performance measures. In fact, our results suggest that aspects such as student work habits, and even code quality, have little bearing on the student's performance. We are eager to have other researchers suggest novel measures which, contrary to our efforts, contain substantial information about productivity, grades, or performance.

Acknowledgments

We thank Karen Reid, Michelle Craig, and Eleni Stroulia for helpful discussions about the data and analysis tools. STR is supported in part by the Canada Research Chairs program and by the IRIS program of NCE.

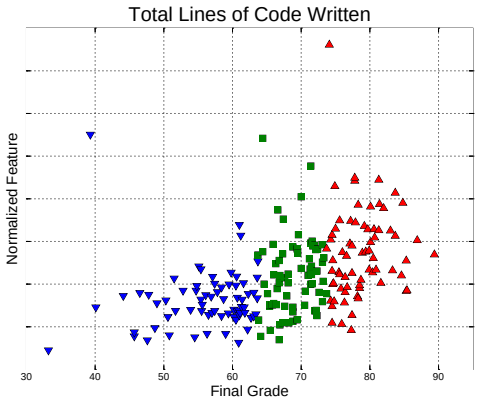

Figure 3: Final grade versus normalized lines of code the student wrote. The correlation visible in this graph between grade and lines of code is as strong as the correlation between grade and any other complex feature we were able to find.

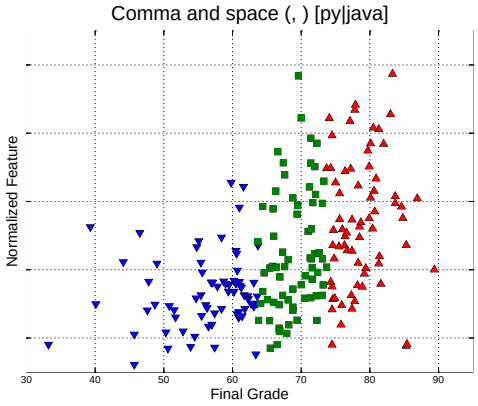

Figure 4: The only other feature to show significant correlation with grade; the number of times a space followed a comma.

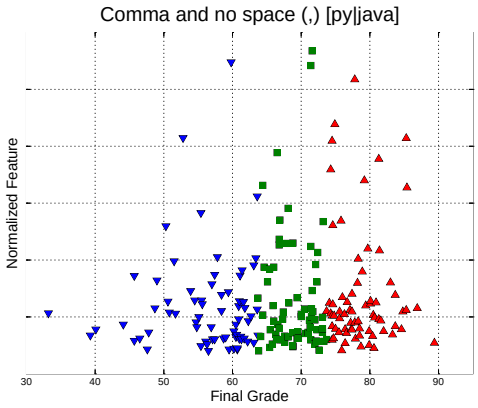

Figure 5: The compliment of the above feature is number of times a comma appears without a subsequent space. From the graph (and the mutual information calculations bear this out) there is very low correlation with grade.

5. REFERENCES

[1] CVS. `http://www.cvs.org/`.

[2] PMD: A style checker. `http://pmd.sourceforge.net/`.

[3] ViewCVS. `http://viewcvs.sourceforge.net/`.

[4] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. In *IEEE Transactions on Software Engineering*, volume 26, July 2000.

[5] Y. Liu, E. Stroulia, K. Wong, and D. German. Using CVS historical information to understand how students develop software. In *Proc. International Workshop on Mining Software Repositories (MSR04), Edinburgh*, 2004.

[6] T. Zimmermann and P. Weibgerber. Preprocessing CVS data for fine-grained analysis. In *Proc. International Workshop on Mining Software Repositories (MSR04), Edinburgh*, 2004.

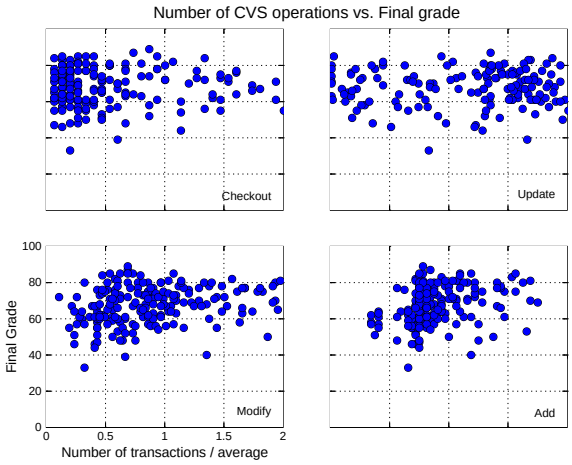

Figure 6: Final grade versus normalized frequency of transactions of various types. (Normalization was done by dividing out the mean.) Each circle represents a single student. The vertical position of the circle in each panel gives the student's final grade in the course while the horizontal position represents a normalized number of transactions of a particular type. Visually, there is no strong correlation present between any of these frequencies and grades; this is confirmed quantitatively by the mutual information analysis in figure 1.

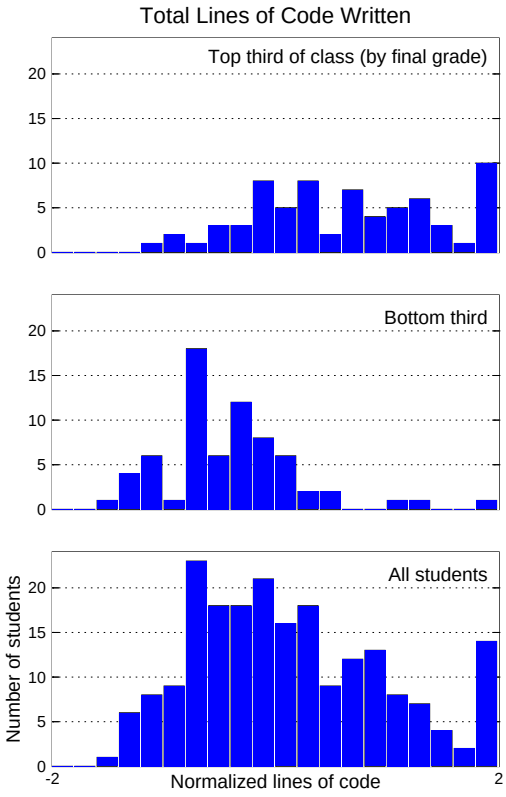

Figure 7: Histograms of (normalized) lines of code written, for the top third (by grade), bottom third, and entire class of 207 students. Visually, it can be seen that students who write more code are more likely to be achieve high grades. This feature was the top scoring predictor of grade in our mutual information analysis and is a statistically significant (though weak) predictor of high grade at the $p = 0.05$ level.