

Detecting Gaming the System in Constraint-Based Tutors

Ryan S.J.d. Baker¹, Antonija Mitrović², Moffat Mathews²

¹ Department of Social Science and Policy Studies, Worcester Polytechnic Institute
100 Institute Road, Worcester MA 01609, USA
rsbaker@wpi.edu

² Intelligent Computer Tutoring Group, Computer Science and Software Engineering
University of Canterbury, New Zealand
{tanja.mitrovic, moffat.mathews}@canterbury.ac.nz

Abstract. Recently, detectors of gaming the system have been developed for several intelligent tutoring systems where the problem-solving process is reified, and gaming consists of systematic guessing and help abuse. Constraint-based tutors differ from the tutors where gaming detectors have previously been developed on several dimensions: in particular, higher-level answers are assessed according to a larger number of finer-grained constraints, and feedback is split into levels rather than an entire help sequence being available at any time. Correspondingly, help abuse behaviors differ, including behaviors such as rapidly repeating the same answer or blank answers to elicit complete answers from the system. We use text replay labeling in combination with educational data mining methods to create a gaming detector for SQL-Tutor, a popular constraint-based tutor. This detector assesses gaming at the level of multiple-submission sequences and is accurate both at identifying gaming within submission sequences and at identifying how much each student games the system. It achieves only limited success, however, at distinguishing different types of gaming behavior from each other.

Keywords: gaming the system, educational data mining, machine learning

1 Introduction

In recent years, increasing attention has been paid to developing educational software which can recognize and adapt to when students game the system. A student games the system when they attempt to succeed in an educational task by systematically taking advantage of properties and regularities in the system used to complete that task, rather than by thinking through the material (cf. [4]). Detectors of gaming the system have been developed through both educational data mining/machine learning methods [5, 7, 9, 28] and through knowledge engineering approaches [1, 10, 16, 25]. These detectors have then been incorporated into interventions shown to reduce gaming and improve learning (cf. [2, 3]). However, despite the broad range of types of educational software where gaming the system has been observed (e.g. [11, 15, 23, 26, 29]), past detectors of gaming the system have been designed for fairly similar types of educational software, where students enter simple answers (a number, a word, or selecting from a set of options) with the problem-solving process reified into individual steps (though sometimes the reification only occurs after an incorrect

answer (cf. [28])). In addition, in these educational software systems, gaming behavior has consisted predominantly of systematic guessing and abuse of multi-level hints (with the exception of [10], where gaming behavior consisted of near-instantaneous responses to multiple-choice questions given a single time). Many other gaming behaviors have been reported in other educational software packages (cf. [11, 21]).

If gaming detection can be applied to a broader range of educational software, we can extend the benefits of gaming intervention to more topics and a greater number of students. In this paper, we study the feasibility of developing gaming detection for SQL-Tutor [19], a constraint-based tutor with design substantially different than environments for which gaming detection has previously been developed, and where students correspondingly engage in different gaming behaviors. Within SQL-Tutor, the “inner-loop” of individual cognitive steps in the problem-solving process is not reified or responded to [27]; complete solutions are analyzed for correctness, making for a different gaming detection challenge than in previous work.

2 Learning Environment and Gaming Behaviors

SQL-Tutor is a constraint-based tutor that assists university-level students in acquiring the knowledge and skills necessary to create SQL queries [19]. It is a mature intelligent tutoring system, designed as a practice environment with the prerequisite that students be previously exposed to the SQL concepts in lectures. Students submit solutions which are sent to the student modeller for analysis. The student modeller identifies any errors and updates the student model accordingly to reflect the student’s progress within the domain. To check the correctness of the student’s solution, SQL-Tutor evaluates the student’s solution against domain knowledge, represented as a set of 700 constraints. A preferred solution, entered by the instructor, aids in this evaluation process while still allowing for novel correct solutions from the student. A constraint consists of a relevance condition C_r , which checks whether the constraint is appropriate for a particular student’s solution, and a satisfaction condition, C_s . A solution is correct if it satisfies the satisfaction conditions of all relevant constraints.

Once the student’s solution is evaluated, the student model passes information to the pedagogical module which generates the appropriate feedback. If any constraints are violated, SQL-tutor will provide feedback on them. In the case where the solution is correct or the student requires a new problem to work on, the pedagogical module uses the information from the student model to select an appropriate problem.

SQL-Tutor provides feedback on demand only, when the student submits the solution. The system offers six levels of feedback, differing in the amount of detail provided to the student. On the first attempt, the system only informs the student whether the solution is correct or not. All other feedback levels provide feedback on errors. The second level (*Error Flag*) points to the part of the solution that is incorrect. The third level (*Hint*) provides a description of one error, pointing out where exactly the error is, what constitutes the error (performing blame allocation) and referring the student to the underlying domain principle that is violated (revising student’s knowledge). The hint message comes directly from the violated constraint.

The automatic progression of feedback levels ends at the hint level; to obtain higher levels of feedback, the student needs to explicitly request them. For example, the student can ask for the hint message for all violated constraints (*All Errors*), a partial solution (showing the correct version of one part of the solution that is wrong), or a complete solution for the problem.

Within SQL-Tutor, several types of gaming behavior can be observed. As in many other intelligent tutors (cf. [1, 5, 16, 28]), systematic guessing is one way for students to game the system and complete problems. Figure 1 presents a sequence of submissions extracted from a student's interaction with SQL-Tutor, in which the student appears to be repeatedly guessing in order to get the answer without having to think through why the answer should be what he/she is typing in. The student

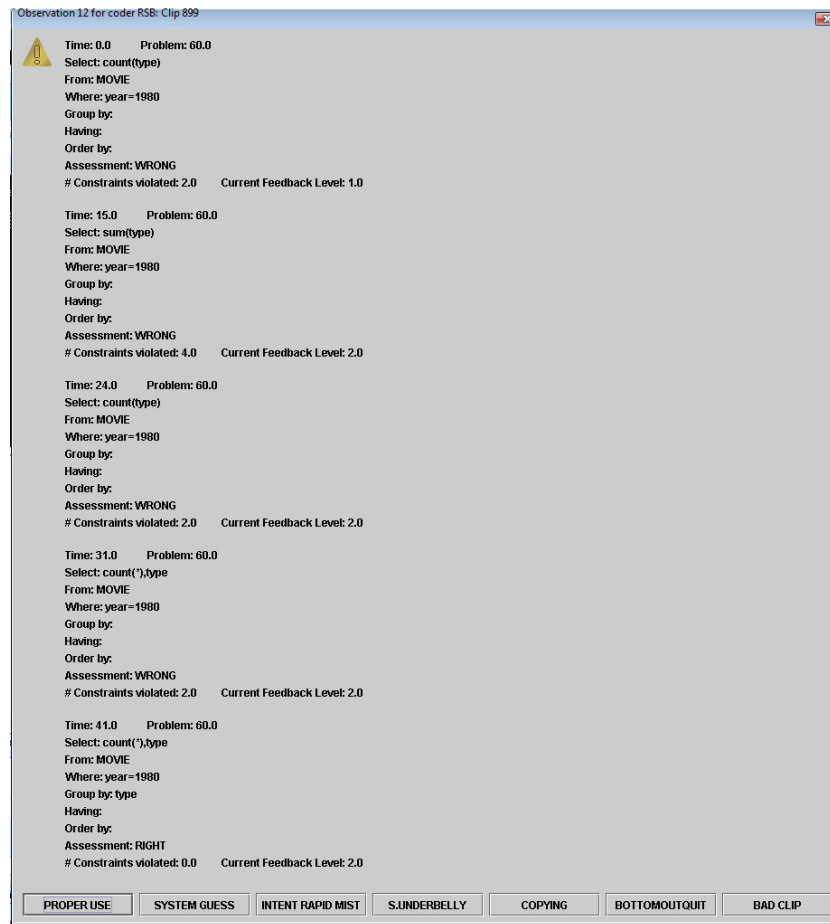


Fig. 1. A "text replay" of student interaction with SQL-Tutor showing systematic guessing.

submitted an initial solution for problem 60, and after being informed that the solution is wrong, simply changed the function used in the SELECT clause. After being informed that the modified solution is still wrong, the student converted back to the previous solution.

However, as discussed, SQL-Tutor handles hints in a different way than many other intelligent tutoring systems, and hence, hint abuse occurs differently. Notably, hint abuse is replaced by requesting the full solution (feedback level 5), reading it and then quitting the problem (BOTTOM-OUT QUIT, illustrated in Fig. 2), a behavior also reported in [24].

Sometimes students ask for full solutions, and simply copy them (COPYING) into their own solution which they then submit. It has been previously found that students who consistently use high-level help in SQL-Tutor frequently employ a “guess then copy” strategy [17]. It is worth noting that both of these strategies could be followed by self-explaining after reading the answer (cf. [25]). However, within SQL-Tutor’s

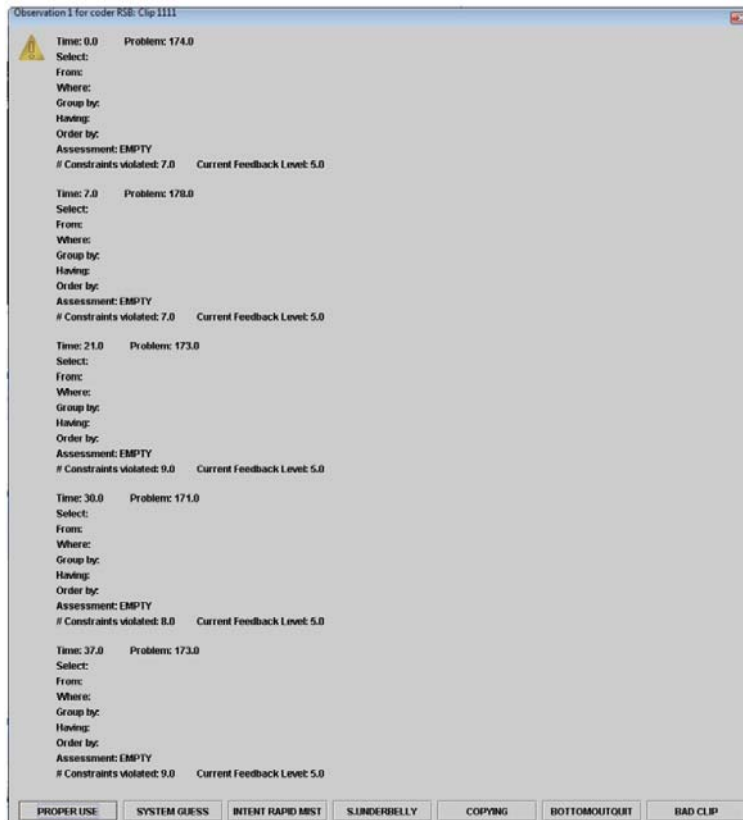


Fig. 2. An example of the “BOTTOM-OUT QUIT” behaviour.

behavior logs, it is difficult to distinguish self-explanation from gaming, due to the lack of step-level reification. Other common gaming behaviors include intentional entering the same answer or a blank answer. In previous analyses within SQL-Tutor, these attempts, labeled as requests for help, have been manually distinguished from valid forms of help-seeking [17]. Intentional rapid mistakes have also been seen in the Andes learning environment [21]. Some students alternatively game the system through “soft underbelly” strategies, repeatedly switching problems to try to find an easier problem to work on.

3 Data and Data Labeling

Data was collected from 61 students using SQL-Tutor in 2003 as part of an introductory database course. The goal of the 2003 study was to investigate two different problem-selection strategies [20]. The students used SQL-Tutor over multiple days ($M=2.4$, $SD=1.8$) in sessions of varying length, up to three hours long. Students completed an average of 19.3 problems ($SD=18.0$), attempting to answer each problem an average of 3.7 times ($SD=1.7$). The total data set consisted of 4428 attempts to solve problems. It should be noted that the number of solution attempts per problem can be expected to be significantly lower in SQL-Tutor than in many other intelligent tutors, as SQL-Tutor requires complete answers (e.g. complete SQL queries) for each problem, rather than reifying each problem step individually. Hence, our data set consisted of roughly an order of magnitude less data per student-time than has been used in previous detectors of gaming behavior trained or developed in more reified intelligent tutors (cf. [5, 8, 16, 28]).

Log files were labeled via text replays with reference to whether the student was gaming [6, 7]. Text replays represent a segment of student behavior from the log files in a textual (“pretty-printed”) form. A sequence of actions of a pre-selected duration (in this case, sets of 5 problem-solving attempts) is shown in a textual format that gives information about the actions and their context. In the portion of a text replay shown in Figures 1 and 2, the coder sees each action’s time (relative to the first action in the clip), the problem number, the input entered, how the system assessed the action (correct, incorrect), the number of constraints violated, and the current feedback level. The coder can then choose one of a set of behavior categories. Text replays provide limited information on student behavior, and require that the coder understand the user interface, in order to interpret contextual information. However, text replays offer several advantages: text replays can be classified extremely quickly (between 9 and 40 seconds per label (cf. [6, 7])), achieve acceptable inter-rater reliability [6], can be generated automatically from existing log files, agree well with assessments generated automatically by gaming detectors trained on field observation data [6], and have been previously used to train detectors of gaming the system [7].

Text replays were conducted by two labelers (the second and third authors) in two rounds. In the first round, a small set of text replays were conducted and inter-rater reliability was poor – Cohen’s Kappa was 0.38 in differentiating gaming behavior from non-gaming behavior and 0.27 in distinguishing all behaviors from each other (e.g. treating different gaming behaviors as separate categories). After the first round,

the two labelers recoded the observations where they had disagreed together and discussed their interpretation of the categories, and then conducted additional labeling. In the second set of text replays, Cohen's Kappa was 0.80 in differentiating gaming behavior from non-gaming behavior and 0.68 in distinguishing all behaviors from each other. In total, a randomly selected 637 sequences of student behavior were coded, accounting for 2297 of the problem-solving attempts.

According to the labels, 38% of the sequences of behavior within SQL-Tutor involved one or more of the categories of gaming the system. This is substantially more gaming than has been seen in previous observational or text-replay based research on the frequency of gaming the system in other intelligent tutors (cf. [4, 7, 13, 28]). This finding is likely to be at least partially due to differences in the logs – while the labeling method was identical to the one used in [7], the different grain-size of the logs likely altered the proportion of gaming observations. For instance, non-gaming behavior during successful intermediate steps of problem-solving is not reified in SQL-Tutor, and thus would not show up in the logs. By contrast, if gaming behavior occurred at the same point in the problem-solving process, it would appear in the logs.

However, it is not clear that the difference in logs accounts for all the difference in gaming frequency; in particular, most of the gaming behavior seen in SQL-Tutor consists of behaviors not seen in the previously studied learning environments. Within SQL-Tutor, requesting the answer and then copying it in, and requesting the answer and then quitting were the most common categories of gaming behavior (respectively accounting for 49.6% and 33.6% of gaming behavior noted). Intentional rapid mistakes, systematic guessing, and soft underbelly strategies were significantly rarer (accounting for 8.6%, 5.7%, and 2.4% of gaming behavior). This suggests that interface design differences may explain some proportion of the greater gaming observed in SQL-Tutor. In particular, the high frequency of answer-request-based gaming strategies may be due to their relative ease of execution, compared to other gaming strategies. It is also possible that SQL-Tutor's lack of reification increases difficulty, and therefore the motivation to game the system; the relatively brief problem scenarios in SQL-Tutor may also contribute to the incidence of gaming behavior (cf. [8]). Another possible explanation for greater gaming in SQL-Tutor may be the setting of use of SQL-Tutor. Whereas the previous environments studied were utilized in classrooms with teachers and other students present (potentially reducing opportunity to game the system), SQL-Tutor is used by students alone in their dorm rooms, or in computer labs where other students generally are not using SQL-Tutor.

4 Detecting Gaming

Once the text replays were obtained, we developed detectors of gaming the system, differentiating gaming behavior from all other categories of behavior (the following section describes our attempts to detect individual gaming behaviors).

The first step was to distil a set of features of student behavior from the log files, which could potentially be predictive of the choice to game the system. In doing so, we built off of prior efforts to detect gaming the system in Cognitive Tutors [5] and ASSISTments [28]. 40 features were distilled for each action in the log files for use

by the classification algorithms, including features involving details about the action (Was it correct or incorrect? Was it the first attempt at the problem? Was the answer empty or a repeat of the previous answer? What was the current help level?), assessments of student knowledge (probability of acquisition of the constraints involved in the problem according to Bayesian Knowledge Tracing – (cf. [12]); how much progress is the student making towards the correct answer with each problem-solving attempt), information about the time the student took on this step (absolute, and relative to other students), and information about the student's previous interaction behaviors (How often has this student gotten this step wrong in the past?).

Having done this, the next task was to determine the grain size of the models of gaming. Past work with developing detectors of gaming the system from text replay data chose the approach of labeling all actions in a text replay according to the majority label [7]. However, since a set of attempts is labeled as gaming if any gaming at all occurs, this approach explicitly labels some non-gaming actions as gaming, and therefore necessarily reduces the assessed accuracy.

Instead, we decided to detect gaming at the grain-size of the sequence of actions labeled rather than at the grain-size of individual attempts. This approach has the disadvantage of delaying the software's response to gaming by 2-4 problem-solving attempts (until the entire 5 action window has completed), but has the advantage of increasing the precision of the detector and its validation (by better aligning exactly what was labeled with the detection process). To this end, for each labeled sequence we averaged each of the distilled variables together (in a running tutor this could be computed as a running window for each variable).

Detection of gaming was conducted in RapidMiner [18]. All reported validation is batch 6-fold cross-validation, at the student level (e.g. detectors are trained on five groups of students and tested on a sixth group of students). By cross-validating at this level, we increase confidence that detectors will be accurate for new groups of students.

Several algorithms were tried within RapidMiner, including J48 decision trees (as in [7, 28]), step regression (as in [5]), support vector machines, naïve bayes, and bagged decision stumps. We assessed the classifiers using five metrics. First, we used A' [14]. A' is the probability that if the detector is comparing two sequences, one involving gaming and one not involving gaming, it will correctly identify which sequence is which. A' is equivalent to both the area under the ROC curve in signal detection theory, and to W , the Wilcoxon statistic [14]. A model with an A' of 0.5 performs at chance, and a model with an A' of 1.0 performs perfectly. In these analyses, A' was used at the level of action sequences rather than students, a different grain-size than used in [5] or [7]. Second and third, we used precision/recall (at the level of action sequences), as assessments of whether the detector was appropriately balancing between identifying gaming and avoiding false positives. Fourth, we used Kappa (at the level of action sequences). Kappa assesses whether the detector identifies the correct action sequences as gaming, better than chance. A Kappa of 0 indicates that the detector performs at chance, and a Kappa of 1 indicates that the detector performs perfectly. Fifth, we used the correlation between a student's observed frequency of gaming the system and their predicted frequency of gaming the system, to assess whether the detector correctly determines how much each student gamed.

The step regression model produced the best performance overall (full details on other algorithms attempted are not given, due to space limitations). The model is:

$$\begin{aligned} G = & 0.752 + 0.187(\text{correct}) - 0.187(\text{incorrect}) \\ & - 0.752(\text{blank answer}) - 0.345 (\text{repeat answer from previous}) \\ & + 0.121 (\text{number of last 8 attempts that were blank or repeat answers}) \\ & - 0.034(\text{number of times database switched in last 3 problems}) \\ & - 0.012(\text{number of errors on this problem so far}) \\ & - 0.222 (\text{pct. of problems abandoned in last 3 sessions}) \\ & + 0.1(\text{total number of problems abandoned}) \\ & + 0.519(\text{first attempt at current problem}) \\ & - 0.133 (\text{lowest probability known of any constraint relevant to problem}) \\ & + 0.225 (\text{average probability known of all relevant constraints}) \\ & - 0.083 (\text{time taken on action, in SD from average time for all students} \\ & \quad \text{attempting this problem-all attempts}) \\ & - 0.017 (\text{time taken on action, in SD from average time for all students} \\ & \quad \text{attempting this problem-first attempts only}) \\ & - 0.245 (\text{current feedback level}) \\ & + 0.081 (\text{most frequent feedback level for this student}) \\ & - 0.178 (\text{"Where" clause used in correct solution to current problem}) \\ & + 0.044 (\text{average number of attempts needed to reach correct answer on} \\ & \quad \text{current problem, across all students}) \end{aligned}$$

Each feature is averaged across the previous five solution-attempts. The resultant value of G is thresholded, using the original cut-off used during training: values *below* 0.5 denote gaming, values equal to or above 0.5 denote non-gaming.

For this model, A' equaled 0.77, meaning that the detector could distinguish a sequence of behavior involving gaming 77% of the time. Precision = 76% and Recall = 87%, indicated fairly balanced performance, though with some bias in favor of indicating gaming. Kappa = 0.36, indicating that the detector was 36% better than chance at identifying gaming (Kappa for detecting gaming in Cognitive Tutors was previously found to be 0.40, a comparable value). The correlation between a student's observed frequency of gaming the system and their predicted frequency of gaming the system was 0.73, better than previous action-level detectors of gaming behavior in Cognitive Tutors (where $r=0.44$). While this detector was clearly not perfect, this level of accuracy is similar to that of previous detectors used in effective interventions given to gaming students (e.g. [2, 3]).

5 Differentiating Between Types of Gaming Behavior

Having developed a reasonably effective detector of gaming behavior overall, in this section we discuss our attempts to differentiate between types of gaming behavior. We attempted to differentiate gaming behaviors in three fashions: differentiating all categories at once, differentiating all of the gaming categories from each other in a sample not including any data originally labeled as non-gaming, and differentiating each gaming category from all other categories. In all cases, the same cross-validation method was used as above.

In differentiating all categories at once, the most effective algorithm was J48. The step regression approach used above was not possible, due to the multiple output classes. Kappa was a relatively poor 0.164 (correlation cannot be calculated for a

multi-class problem, and calculating multi-class A' for J48 decision trees is highly non-trivial). The model was most accurate at capturing non-gaming behavior (Precision=69%, Recall = 78%). Performance was somewhat above zero at capturing quitting after requesting the answer (Precision = 27%, Recall = 19%) and requesting the answer and copying it in (Precision = 28%, Recall = 27%). Precision and recall were 0 for all other classes.

In differentiating each of the gaming categories from each other (in a data set omitting behavior originally labeled as non-gaming), the most effective algorithm was J48. Kappa was a relatively poor 0.142. The classes best captured were quitting after requesting an answer (Precision = 49%, Recall = 48%) and requesting the answer and copying it in (Precision = 58%, Recall = 63%). The other classes had Precision and Recall under 15%.

In capturing single gaming categories, the most effective algorithm in all cases was J48 (performing better than step regression). The classes best captured were once again quitting after requesting the answer (A' =0.76, Kappa = 0.22, Precision = 36%, Recall = 26%) and requesting the answer and copying it in (A' = 0.61, Kappa = 0.14, Precision = 32%, Recall = 25%). The model was not able to achieve Precision or Recall over zero for any of the other classes.

Considering that the two categories involving requesting the complete answer were predicted better than the other categories, it may be worth separating out the gaming involving answer requests and the gaming not involving answer requests. If we attempt to differentiate answer-request-based gaming from all other behavior, performance is relatively decent, though still not as good as the performance for identifying gaming in general (A' =0.72, Kappa = 0.30, Precision = 55%, Recall = 47%). If we compare gaming involving answer requests to two other categories, gaming not involving answer requests, and non-gaming behavior, Kappa drops to 0.235. The precision and recall for the unified answer-request gaming category is also worse in this case (Precision = 49%, Recall = 43%).

In general, it is worth considering why machine learning was so successful at identifying gaming the system at a general level, but unable (for the most part) to successfully distinguish between types of gaming beyond distinguishing answer requests from other types of gaming. One possibility is that different gaming behaviors, while looking different within the text replays, are actually quite similar in terms of how they appear in our feature set – occurring in similar contexts, and manifesting in similar ways (e.g. lots of rapid errors or help requests). It therefore may be possible to do better at distinguishing types of gaming by improving the feature set, potentially by including a greater breadth of low-level information on the responses, to try to create a closer match between the information used by the labelers and the information used by the machine learning algorithm.

At the same time, it is useful information that the gaming behaviors that were in themselves most easily distinguishable were the two most common gaming behaviors. Hence it may simply be that the other gaming behaviors were not common enough in the data set to form the basis of an effective single-behavior gaming detector. In the whole data set, there were only 13 observations of systematic guessing, 5 observations of soft underbelly strategies, and 17 observations of intentional rapid mistakes, and these behaviors were clustered in small numbers of students. Hence,

replicating with a larger data set may improve detection of some of the rarer gaming behaviors.

Another reason why distinguishing gaming may be difficult is that many clips showed evidence for more than one kind of gaming. Commonly, these clips showed strong evidence for one type of gaming, while having weaker evidence for another type of gaming – for instance, when a student tried two answers quickly and then repeated the second answer multiple times until the system gave the correct answer. In this case, the coders both identified this sequence of actions as intentional rapid mistakes, but it is understandable that the machine learning algorithm might incorrectly label what type of gaming was predominant in this clip.

6 Conclusions

In this paper, we have presented an accurate detector of gaming the system for SQL-Tutor, a constraint-based tutor. This marks the first detector of gaming for an intelligent tutoring system that does not reify student thinking at the “inner loop” of problem-solving performance (e.g. [27]) – where students are only evaluated at the level of complete answers, rather than being evaluated at multiple steps of the problem-solving process. Constraint-based tutors differ from other types of intelligent tutors for which gaming detectors have been developed in other fashions as well: student answers are assessed according to a larger number of finer-grained constraints, and feedback is split into levels rather than an entire help sequence being available at any time, though complete answers are available upon request. Correspondingly, gaming behaviors differ, with high frequency for behaviors such as requesting the answer, copying it in, and quitting.

The detector we present, developed using a combination of text replays and educational data mining/machine learning methods, assesses gaming behavior across sequences of five problem-solving attempts. It can accurately identify both when a student is gaming (action-level $A' = 0.77$, Precision = 76%, Recall = 87%, Kappa = 0.36), and how much each student games the system (correlation = 0.73), performance in line with previous detectors of gaming the system. As such, it should be usable to drive interventions when students game the system (cf. [2, 3]). The detector was also able to differentiate gaming behavior involving directly requesting the answer from other types of gaming behavior, albeit with lower success.

However, detector performance was considerably worse at differentiating between each type of gaming behavior, potentially because gaming behaviors occur in similar contexts and share similarities in their features (e.g. fast actions, requesting the answer). One possibility is that the feature set, in only looking at high-level differences between responses, may be missing more subtle patterns that human coders were able to utilize to identify different types of gaming. It is worth noting that there has been previous success at a different type of differentiation, differentiating the same gaming behaviors in different situations (such as gaming on poorly known skills versus well-known skills; or gaming and then self-explaining) [5, 25]. At the same time, this type of differentiation also might be more difficult to produce in SQL-Tutor, due to the higher-level logging (which occurs because intermediate problem-solving steps are not reified). For instance, Shih and colleagues [25] were able to

differentiate gaming from self-explaining after a bottom-out hint because of the short time until the next reified problem-solving step observed in most cases; this same type of differentiation would not be possible within SQL-Tutor's logs.

However, an alternate possibility is that in this study the rarer categories were poorly identified due to a lack of sufficient training examples. Hence, one potential future step would be to obtain more labels of those categories. In doing this, it might be valuable to sample the data for human labeling in a fashion biased against action sequences which have a high confidence of being non-gaming or involving the two more common gaming behaviors (a process analogous to automated biased sampling procedures – (cf. [22])).

Acknowledgements. The development of SQL-Tutor was supported by the University of Canterbury research grant. We thank all members of ICTG for their support. This research was supported by the Pittsburgh Science of Learning Center (National Science Foundation) via grant “Toward a Decade of PSLC Research”, award number SBE-0836012.

References

1. Aleven, V., McLaren, B., Roll, I., Koedinger, K.: Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor. *Artificial Intelligence and Education*, 16, 101-128 (2006)
2. Arroyo, I., Ferguson, K., Johns, J., Dragon, T., Meheranian, H., Fisher, D., Barto, A., Mahadevan, S., Woolf, B.P.: Repairing Disengagement with Non-Invasive Interventions. In: *Proc. 13th Int. Conf. Artificial Intelligence in Education*, 195-202 (2007)
3. Baker, R.S.J.d., Corbett, A.T., Koedinger, K.R., Evenson, S.E., Roll, I., Wagner, A.Z., Naim, M., Raspat, J., Baker, D.J., Beck, J.: Adapting to When Students Game an Intelligent Tutoring System. In: M. Ikeda, K. Ashley, and T.-W. Chan (Eds.): *Lecture Notes in Computer Science* 4053, 392-401 (2006)
4. Baker, R.S., Corbett, A.T., Koedinger, K.R., Wagner, A.Z.: Off-Task Behavior in the Cognitive Tutor Classroom: When Students "Game The System. In: *Proc. ACM CHI 2004: Computer-Human Interaction*, 383-390 (2004)
5. Baker, R.S.J.d., Corbett, A.T., Roll, I., Koedinger, K.R.: Developing a Generalizable Detector of When Students Game the System. *User Modeling and User-Adapted Interaction*, 18 (3), 287-314 (2008).
6. Baker, R.S.J.d., Corbett, A.T., Wagner, A.Z.: Human Classification of Low-Fidelity Replays of Student Actions. In: *Proc. Educational Data Mining Workshop at the 8th Intl. Conf. on Intelligent Tutoring Systems*, 29-36 (2006)
7. Baker, R.S.J.d., de Carvalho, A. M. J. A.: Labeling Student Behavior Faster and More Precisely with Text Replays. In: *Proc. 1st Int. Conf. Educational Data Mining*, 38-47 (2008)
8. Baker, R.S.J.d., de Carvalho, A.M.J.A., Raspat, J., Aleven, V., Corbett, A.T., Koedinger, K.R.: Educational Software Features that Encourage and Discourage "Gaming the System". In: *Proc. 14th Int. Conf. Artificial Intelligence in Education*, 475-482 (2009)
9. Beal, C.R., Qu, L., Lee, H.: Mathematics motivation and achievement as predictors of high school students' guessing and help-seeking with instructional software. *Journal of Computer Assisted Learning*, 24, 507-514 (2008)
10. Beck, J.: Engagement tracing: using response times to model student disengagement. In: *Proc. 12th Int. Conf. on Artificial Intelligence in Education*, 88-95 (2005)

11. Cheng, R., Vassileva, J.: Design and evaluation of an adaptive incentive mechanism for sustained educational online communities. *User Modeling and User-Adapted Interaction*, 16 (3/4), 312-348 (2006)
12. Corbett, A.T., & Anderson, J.R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4, 253-278 (1995).
13. Gobel, P.: Student Off-task Behavior and Motivation in the CALL Classroom. *International Journal of Pedagogies and Learning*, 4 (4), 4-18 (2008)
14. Hanley, J.A., McNeil, B.J.: The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, 143, 29--36 (1982)
15. Heilman, M., Eskenazi, M.: Language Learning: Challenges for Intelligent Tutoring Systems. In: Proc. Workshop on Intelligent Tutoring Systems for Ill-Defined Domains, Proc. 8th Int. Conf. Intelligent Tutoring Systems (2006)
16. Johns, J., Woolf, B.: A Dynamic Mixture Model to Detect Student Motivation and Proficiency. In: Proc. 21st National Conference on Artificial Intelligence (AAAI-06), 163-168 (2006)
17. Mathews, M., Mitrovic, A.: How Does Students' Help-Seeking Behaviour Affect Learning? In: E. Aimeur, B. Woolf, R. Nkambou (Eds.), *Lecture Notes in Computer Science* 5091, 363-372 (2008).
18. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), 935-940 (2006)
19. Mitrovic, A.: An Intelligent SQL Tutor on the Web. *Artificial Intelligence in Education*, 13(2), 173-197 (2003)
20. Mitrovic, A., Martin, B. Evaluating adaptive problem selection. In: P. De Bra, W. Nejdl (eds) *Lecture Notes in Computer Science* 3137, 185-194 (2004).
21. Murray, R.C., VanLehn, K.: Effects of dissuading unnecessary help requests while providing proactive help. In: Proc. 12th International Conference on Artificial Intelligence in Education, 887-889 (2005)
22. Palmer, C.R., Faloutsos, C.: Density biased sampling: an improved method for data mining and clustering. In: Proc. 2000 ACM SIGMOD Int. Conf. Management of Data, 82-92 (2000)
23. Rodrigo, M.M.T., Baker, R.S.J.d., Lagud, M.C.V., Lim, S.A.L., Macapanpan, A.F., Pascua, S.A.M.S., Santillano, J.Q., Sevilla, L.R.S., Sugay, J.O., Tep, S., Viehland, N.J.B.: Affect and Usage Choices in Simulation Problem Solving Environments. In: Proc. 13th Int. Conf. Artificial Intelligence in Education, 145-152 (2007)
24. Schofield, J.W. *Computers and Classroom Culture*. Cambridge, UK: Cambridge University Press (1995)
25. Shih, B., Koedinger, K., Scheines, R.: A Response Time Model for Bottom-Out Hints as Worked Examples. In: Proc. 1st Int. Conf. on Educational Data Mining, 117-126 (2008)
26. Tait, K., Hartley, J., Anderson, R.C.: Feedback procedures in computer-assisted arithmetic instruction. *British Journal of Educational Psychology*, 43, 161-171 (1973)
27. VanLehn, K.: The Behavior of Tutoring Systems. In: *International Journal of Artificial Intelligence in Education*, 16 (3), 227-265 (2006)
28. Walonoski, J.A., Heffernan, N.T.: Detection and Analysis of Off-Task Gaming Behavior in Intelligent Tutoring Systems. In: In: M. Ikeda, K. Ashley, T.-W. Chan (Eds.): *Lecture Notes in Computer Science* 4053, 382-391 (2006)
29. Wood, H., Wood, D.: Help Seeking, Learning, and Contingent Tutoring. *Computers and Education*, 33, 153-169 (1999)