January 2015

The Hidden Value of Common Wrong Answers

Bohao Li
Worcester Polytechnic Institute

Douglas Selent
Worcester Polytechnic Institute

The Hidden Value of Common Wrong Answers

An Interactive Qualifying Project Report:
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements of the
Degree of Bachelor of Science
By

_____

Bohao Li

In collaboration with

_____

Douglas Selent

Date: December 19, 2014

Advisor:

_____

Professor Neil Heffernan

Contents

Abstract

In this paper, we investigate the value of common wrong answers. We carry three questions. First, how do we define common wrong answers? Second, how common are the common wrong answers following our definition? Third, by introducing common wrong answers into model building, can we achieve better models in predicting next problem correctness? Next problem correctness stands for the probability that a student will get the next problem in the same assignment correct after finishing the current problem. To answer the first question, we proposed a definition for common wrong answers. To answer our second question, we examined the prevalence of common wrong answers within our data sets. To answer our third question, we built two tabling models. The first model is our control model. This model makes predictions on next problem correctness based on three types of student responses, a response where the student give a correct answer (correct response), a response where the student ask for a hint (hint response) and a response where the student give a wrong answer (wrong answer response). The second model is our experiment model, named Li, Selent & Heffernan's Interesting Common Wrong Answers Model (ICWAs Model). The ICWAs model extends upon the control model by giving each common wrong answer its own prediction. We then compared the results from both models and verified that common wrong answers do not bring reliable improvements to predicting next problem correctness.

Keywords: tabling; common misconceptions; predictive modeling;

Introduction

Student incorrect responses were first studied in 1978 by John Seely Brown and Richard R. Burton[1]. They manually analyzed student incorrect responses for multi-digit subtraction problems. Incorrect processes or "bug"s performed by the student attributed to these incorrect responses. From this analysis, they created a procedural network, where a skill is broken down into sub-skills. Their main goal was to construct a database of all the possible buggy procedures with their series of "BUGGY" programs that helped to identify bugs [3]. Brown and VanLehn extended this work with their introduction of "Repair Theory" [2]. The idea of Repair Theory is that when a student realizes that he/she has performed a buggy operation he/she will attempt to repair the bug. This work looked at how bugs are caused and what bugs can be predicted.

Several researchers explored the area of finding common incorrect processes in 1990's summarized in [7]. Various machine-learning algorithms in different systems attempted to predict student misconceptions and react appropriately. More recent algorithms have been made to automatically discover these buggy rules. One of the most recent examples is a machine-learning algorithm developed in [6]. In their work, a machine-learning algorithm was developed to automatically discover all buggy rules for a given set of problems generated by the same template. This algorithm works by first taking the input symbols of the problem (the numbers in the problem), a set of basic operations (addition, subtraction, multiplication, and division), and all the student incorrect responses for all problems generated by a single template. Next, the algorithm derives all possible incorrect processes for all the incorrect responses and generalizes the incorrect response across all problems generated by a given template. Finally, the incorrect processes are assigned to the most likely generalized incorrect processes. The output of the algorithm is the machine learned process for the incorrect response (how the student arrived at their incorrect answer), as well as a percentage breakdown for all the incorrect processes. Although the method has a few weaknesses in terms of computation time and computer memory, it is a sufficient solution to the problem of finding bugs.

Despite such a large amount of past work done on predicting student incorrect responses, there is little work done to use these incorrect response to predict future performance or correctness on the next problem for fill-in problems. The National Council on Measurement in Educational Measurement (NCME) community has come up with methods that weight incorrect responses differently. For example, work done by DeMars shows that polytomous models

(weighting the incorrect responses differently) for multiple-choice questions predict better than dichotomous models (weighting all incorrect responses the same) [4]. Our contribution in this paper is to use the idea of weighting incorrect responses differently that is some wrong answers indicate very poor knowledge, while other wrong answers are associated with very high knowledge and apply it to fill-in problems. DeMars notes that "The polytomous models, which weighted the distractors differentially, yielded small increases in reliability compared to their dichotomous counterparts." We believe there is value to this and hope to find larger improvements with fill-in questions as opposed to the multiple-choice questions DeMars used.

The goal of this paper is to see how well we can predict student performance on the next question by knowing the commonality of their previous incorrect responses. In this paper, we hope to answer three questions. (1) How do we define common wrong answers? (2) How common are the common wrong answers following our definitions? (3) Can we build better models in predicting next problem correctness using common wrong answers?

To answer these questions, we use one dataset described in the data section, define common wrong answers and examine their commonality. We then build two tabling models, a control model and an experiment model. We compare the performance on both models and test if they perform reliably different from one another. After the initial experiments, we conduct more experiments to test if our model generalize across students.

Data

For our experiments, we use one skill builder dataset of fill-in question responses on mathematical questions. The data ranges from years 2008-2014 and grades 4-12 from the ASSISTments online tutoring system. ASSISTments is a tutoring system mostly used for mathematics in grades 4-12. Most of the users in the system are in the United States with a large number of students located in or near Massachusetts. Students use ASSISTments on classwork and homework, which may be done with or without the use of a paper copy. The system typically provide instant feedback to the student upon answering the problems, so they know immediately whether they have answered correctly. Students cannot skip any problems and must answer correctly to continue.

A skill builder is a set of problems where a student must get a certain number of problems correct in a row (usually three) in order to complete. A skill builder typically consists of a bank of 50-100 possible questions that are randomly drawn from and given to the student. One or more templates generate these questions. For example, a template problem for the skill "Order of Operation", looks like "a + b * c". This template generates the problem instances "1 + 2 * 3", and "5 + 4 * 9" as well as several other similar problems by substituting different numbers in for the variables 'a', 'b', and 'c' [5]. The reason why we chose to focus on data from skill builders is that this data is more realistic and less subject to noise, which is a common problem in online tutoring systems. Skill builders are used by several teachers and are not specific to a certain demographic. Due to the randomized questions, skill builders are also less subject to cheating by students. Students are given correctness feedback, on whether they got the problem right or wrong, after submitting an answer for a problem in. Our selected dataset contains 635,443 rows with 35,928 students, 14,393 problems and 9,960 assignments. This dataset, along with the code for our experiments and the models we have built, is available at https://sites.google.com/site/commonwronganswerscodeanddata/my-documents

Common Wrong Answer Definition

The word ‘common’ in ‘common wrong answers’ suggests that our common wrong answers should contain a minimum number of student responses and cover a certain percentage of the responses covered by all wrong answers in the same problem. Thus, we come up with the following definition for common wrong answers:

1) Wrong answers that contain more than 19 responses. We will refer to this constraint later in the paper as minimum number of responses.
2) Wrong answers that cover more than 10% of all student responses covered by wrong answers in the same problem.

In our paper, we want to focus on the common wrong answers that actually contribute to making better predictions on next problem correctness – common wrong answers who distinguish themselves from all other wrong answers. We define these common wrong answers as Interesting Common Wrong Answers (ICWA). In addition to satisfying the general common wrong answer definition, ICWAs also need to satisfy the following requirement:

3) Wrong answers whose next problem correctness differ from the averaged wrong answer next problem correctness of the same problem by at least 0.18. (e.g. If the averaged wrong answer next problem correctness of problem A is 0.50, a wrong answer may be considered a common wrong answer if its next problem correctness is either above 0.68 or below 0.32. We introduce this constraint to ensure that we consider only common wrong answers that help make better predictions.) We will refer to this constraint later in the paper as minimum difference.

In this case, our values for minimum number of responses and minimum difference are arbitrary. We will refer to the above definition of Interesting Common Wrong Answers as the default definition of ICWAs. We expect to find two kinds of ICWAs in our dataset. The firsts are positive ICWAs suggesting good student knowledge/skills; the seconds are negative ICWAs suggesting very poor student knowledge/skills. In our dataset, out of 14,393 problems, 63 problems contain default ICWAs. Below are three examples of these 63 problems with Interesting Common Wrong Answers.

Problem ID	132104		
Problem Context	Simplify the following: (8 - 4y) - (3y - 7) In order to type your answer in you must do just like you do with a graphing calculator: -no spaces between factors and operations; -don't use * for multiplication, For example: 4x-7y		
Correct Answer	15 - 7y		
All Wrong Answers	Next Problem Correctness	Number of Responses	
	0.3819	233	
Common Wrong Answer 1 (positive ICWA)	Next Problem Correctness	Answer Text	Number of Responses
	0.5925	"1-7y"	27
Common Wrong Answer 2	Next Problem Correctness	Answer Text	Number of Responses
	0.3461	"1-1y"	26
All other Wrong Answers	Next Problem Correctness	Number of Responses	
	0.3555	180	
Analysis	Observing the two common wrong answers, the first common wrong answer has a much higher next problem correctness than the averaged wrong answer next problem correctness and the second common wrong answer. The reason is that students giving the first common wrong answer are partially correct, computing the "-7y" part of the expression whereas students who giving the second common wrong answer are completely wrong. Only the first common wrong answer is an interesting common wrong answer.		

Table 1 shows the details of problem 132104, which holds a positive ICWA

Problem ID	34444
Problem Context	What is 12 - (-14)?
Correct Answer	26

All Wrong Answers	Next Problem Correctness	Number of Responses
	0.4878	205
Common Wrong Answer 1 (negative ICWA)	Next Problem Correctness	Answer Text	Number of Responses
	0.2972	"-26"	37
Common Wrong Answer 2	Next Problem Correctness	Answer Text	Number of Responses
	0.5333	"2"	60
Common Wrong Answer 3	Next Problem Correctness	Answer Text	Number of Responses
	0.5000	"-2"	86
All other Wrong Answers	Next Problem Correctness	Number of Responses
	0.5454	22
Analysis	Compared to other common wrong answers, our Interesting Common Wrong Answer "-26" has a much lower next problem correctness. Common wrong answer "-2" and "2" can be explained by students mistaking the expression for "12 -14" where "-26" is more difficult to explain. The negative ICWA in this problem suggests poor substraction skills.

Table 2 shows the details of problem 34444, which holds a negative ICWA

Problem ID	403245
Problem Context	Multiply 0.72 and 0.57, rounding the answer to the nearest thousandth.
Correct Answer	0.410
All Wrong Answers	Next Problem Correctness	Number of Responses
	0.6279	43
Common Wrong Answer 1 (positive ICWA)	Next Problem Correctness	Answer Text	Number of Responses
	0.8421	"0.41"	19
Uncommon Wrong	Next Problem Correctness	Number of Responses

Answers	0.4583	24
Analysis	In this problem, the students giving the positive ICWA actually performed the correct calculation but they failed to round the answer to the nearest thousandth. Careless mistakes resulted this ICWA and it suggests good student knowledge.	

Table 3 shows the details of problem 34444, which holds a positive ICWA

It is clear that in these sample problems, common wrong answers contribute to our insight to student knowledge and may help better predict students' next problem correctness. The 63 problems with ICWAs makes up only a very small portion of the 14,393 problems in our entire dataset. However, many of the 14,393 problems in our data set do not have enough student responses to have ICWAs. Of the 63 questions that contain default ICWAs (minimum difference = 0.18, minimum number of responses = 19), 48 of which contain more than 100 student responses with wrong answers. Of the 14,393 problems contained in our dataset, only 208 of which contain more than 100 student responses with wrong answers. Among all questions with more than 100 wrong responses, 23% of which contain default ICWAs. From this finding, we can see that the ICWAs model will likely cover more problems if more students have worked on the problems in our dataset. If all 14,393 problems have more than 100 wrong responses, we can reasonably expect over 3,000 problems with default ICWAs. Given these observation, we push forth to designing an experiment investigating whether ICWAs reliably help predict next problem correctness.

Models

A tabling model provides a mapping from data to predictions based on the attributes of the data. This generates a probability table to use for predicting. Tabling has been applied in past research of Wang et al [8]. Their tabling method provided a complement to the Knowledge Tracing model by using past response sequences to predict future responses. To demonstrate the positive effect of common wrong answers, we introduce two tabling models in predicting next problem correctness.

Control Model

Our control model contains three next problem correctness predictions for each problem in the dataset based on the students' first responses, which can be correct responses, wrong responses and requests for hints. The prediction values are calculated by averaging all matching responses' next problem correctness in the training set.

ICWAs Model

Our ICWAs model expands on our control model by splitting the wrong response category in the control model into ICWA categories, each containing a prediction for students who make the exact ICWA, and an all other wrong answer category, containing a prediction for students who make all other wrong answers. The prediction values are calculated by averaging all matching responses' next problem correctness in the training set.

An example of each tabling method is shown in tables 1-2. The example tables display only one problem where the actual tables contain all problems that can be used in predicting next problem correctness in the training set.

Problem ID	Answer	Prediction	Data Points
…	…	…	…
248694	CORRECT	0.7260	73
248694	HINT	0.2857	28
248694	Wrong Answer	0.3819	233
…	…	…	…

Table 4 shows an example of what part of control model looks like. It is the simpler than the ICWAs model.

Problem ID	Answer	Prediction	Data Points
…	…	…	…
248694	CORRECT	0.7260	73
248694	HINT	0.2857	28
248694	“1-7y”	0.5925	27
248694	ALL OTHER WRONG ANSWERS	0.3555	206
…	…	…	…

Table 5 shows an example of what part of ICWAs model looks like. It has broken the wrong answer category in the control model into common and uncommon wrong answers with each common wrong answer having its own prediction value

Experiments

In conducting our experiment, we have separated our dataset into the training set and the test set. The training set consists of the responses from students whose last digit in their user ids are not 1 nor 6, making up about 80% of the whole dataset. The test set consists of responses from students whose last digit in their user ids are 1 or 6, making up about 20% of the whole dataset.

We are training our models on student-assignment bases. For example, problem B will only be considered as the next problem of problem A only if the same student worked on problem B after problem A with both problems in the same assignment. Because of this limitation, our models cannot predict student's first problem correctness in each assignment. After running our models on the test set, we are able to gather 83,267 predictions made by both models. We use RMSE (Root Mean Squared Error) and R square to evaluate the performance of both models.

	Control Model on All Responses	ICWAs Model on All Responses
RMSE	0.4710	0.4703
R square	0.1105	0.1132

Table 6 shows the comparison between the control model and the ICWAs model in terms of RMSE and R square on all responses

As measured by RMSE, the ICWAs Model does not perform reliably better than the Control Model. This is understandable because common wrong answers cover only a narrow portion of the whole dataset. Of the 83,267 predictions made by the ICWAs Model, only 178 predictions are made using the reliable common wrong answers matching our criteria, consisting of only 0.2% of all predictions. From here, we may observe that while our sample problems suggest that common wrong answers can potentially help making better predictions on next problem correctness, they cover too few student responses to have an obvious impact when measured using the whole test set. Given this issue, we narrow our comparisons to just the 178 predictions made by the 63 problems with ICWAs using ICWAs. With the scope narrowed down, we start to observe reliable improvements.

	Control Model on the 178 responses	ICWAs Model on the 178 responses
RMSE	0.4946	0.4643
R square	0.0202	0.1364

Table 7 shows the comparison between the control model and the ICWAs model in terms of RMSE and R square on ICWA predictable responses

Although promising, we still need to verify if other student groups show the same improvement. The previous results are from the test set with student ids ending with 1 or 6 and the training set with student ids that aren't. We conducted experiments on other training/test sets and the results show less promise.

Training Set Student id Last Digit	Test Set Student id Last Digit	Control Model RMSE	ICWAs Model RMSE	Number of Predictions Made
Not 0 nor 5	0 or 5	0.493269989	0.529239096	181
Not 1 nor 6	1 or 6	0.494623739	0.464383894	178
Not 2 nor 7	2 or 7	0.497762406	0.49546321	201
Not 3 nor 8	3 or 8	0.491225063	0.492849082	166
Not 4 nor 9	4 or 9	0.499117646	0.494227533	212

Table 8 shows the comparison between the control model and the ICWAs model in terms of RMSE ICWA predictable responses with different student groups considered

The averaged RMSE from the Control Model is 0.495449 and the averaged RMSE from the ICWAs Model is 0.495341. For the ICWAs model, the RMSE results are calculated using only the predictions made by Interesting Common Wrong Answers. While the ICWAs model perform well on one group of students (id last digit 1 or 6), the improvement fail to generalize to all students.

Analysis

Given that there are indeed problems in our dataset that contain meaningful common wrong answers, ICWAs failing to provide any improvements to predicting next problem correctness might have the following causes:

1. Common wrong answers do not hold enough predictive power as correct and hint responses to distinguish themselves from all other wrong answers. The key question we investigated is if a student makes an Interesting Common Wrong Answer, is he/she more likely to get the next problem correct/incorrect than if he/she makes one of all other wrong answers? Our experiments suggest that the answer is no. In general, for fill-in problems, a wrong answer is just like any other wrong answers and it does not matter if it is interestingly common.
2. Student responses are scattered among a huge amount of problems in our dataset and are not concentrated enough to produce ICWAs. Our ICWA definition require only a minimum of 19 responses. A dataset large enough to produce a good number of ICWAs with more than 100 student responses might yield better results.

Conclusion

We set out to investigate the possibility of improving our ability to predict student performance depending on whether they provided a common wrong answer to a previously related question. We were inspired by work done by DeMars, who showed that polytomous models predict performance better than dichotomous models on multiple-choice questions. We started by defining common wrong answers and Interesting Common Wrong Answers. We then identified a few problems showing that ICWAs make a difference in predicting next problem correctness. We then conducted experiments showing ICWAs bring no improvements to predicting next problem correctness. Our common wrong answers do not distinguish from all other wrong answers in their ability to predicting next problem correctness.

Contributions

We are the first people to our knowledge that have shown that paying attention to common wrong answers can not provide value to accessing student learning. We have provided solid examples showing that our Interesting Common Wrong Answers do not help make better predictions in next problem correctness.

The result may provide insight to future researchers studying common wrong answers. Common wrong answers can suggest student knowledge, but they do not help predicting next problem correctness. Future researches involving common wrong answers can tackle impacts other than next problem correctness.

References

Brown, John Seely, & Burton, Richard R.. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive science 2.2*, 155-192.

Brown, John Seely, & VanLehn, Kurt. (1980). Repair theory: A generative theory of bugs in procedural skills. Cognitive science 4.4, 379-426.

Burton, R. R. Diagnosing bugs in a simple procedural skill. In D. H. Sleeman, & J. S. Brown (Eds.), Intelligent tutoring systems, 157-183. New York: Academic Press.

DeMars, C. E. (2008). Scoring multiple choice items: A comparison of IRT and classical polytomous and dichotomous methods. Paper presented at the National Council on Measurement in Education. New York, NY. Retrieved Oct 29, 2014 from https://www.jmu.edu/assessment/CED%20NCME%20Paper%2008.pdf

Razzaq, L., Patvarczki, J., Almeida, S., Manasi,V., Feng, M., Heffernan, N. T., & Koedinger, K. R. (2009). The Assistment Builder: Supporting the life cycle of tutoring system content creation. *IEEE Transactions on Learning Technologies*, Special Issue on Real-World Applications of Intelligent Tutoring Systems. 2(2), 157-166

Selent, D., & Heffernan, N. T. (2014). Reducing Student Hint Use by Creating Buggy Messages from Machine Learned Incorrect Processes. (pp. 674-675). In Stefan Trausan-Matu, et al. (Eds) *International Conference on Intelligent Tutoring 2014.* LNCS 8474. Retrieved Dec 22, 2014 from https://drive.google.com/viewerng/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxzZWxlbnRpdHMyMDE0fGd4OjFjZDU5YzllYjk3ZDdmNg

Sison, R., & Masamichi S. (1998). Student modeling and machine learning. *International Journal of Artificial Intelligence in Education (IJAIED) 9*, 128-158.

Wang, Q., Pardos, Z. A., & Herffernan, N. T. (2011). Response Tabling-A simple and practical complement to Knowledge Tracing. In *KDD workshop*. Retrived Oct 29th 2014 from https://pslcdatashop.web.cmu.edu/KDD2011/papers/D-kddined2011.pdf

Appendix

Python code for this project:

```
"""
	File to generate the tabulation results given the problem log data

	@author Bohao Li
"""

import csv
import sys
import math

from sets import Set

#
# global variables, should be maintained as constants
#

#
# set the indexes of columns in our data set
#
iorder_id = ord('A') - ord('A')
iuser_id = ord('C') - ord('A')
iskill_id = ord('Q') - ord('A')
iassignment_id = ord('B') - ord('A')
iproblem_id = ord('E') - ord('A')
icorrect = ord('G') - ord('A')
isequence_id = ord('L') - ord('A')
ihint_count = ord('U') - ord('A')
ianswer_text = ord('Z') - ord('A')

#
# student id % 5 == 1 will be in the test set, the rest will be in the training set
#
MOD_GROUP = 1
MOD_NUM = 5

#
# default min difference and min response to our cwas
#
MIN_RESPONSES = 19
MIN_DIFFERENCE = 0.18

#
# Problem class to hold all info to a certain problem
#
class Problem:
	def __init__(self, problem_id, sequence_id):
		self.id = problem_id
		self.sequence_id = sequence_id
		self.answers = {}
```

```
                self.total_uncommon = 0
                self.total_common = 0
                self.total_entries = 0
                self.uncommon_next_correct = 0
                self.uncommon_next_incorrect = 0
                self.common_next_correct = 0
                self.common_next_incorrect = 0
                self.wrong_answer_correct_rate = 0

#
# Problem answer class that holds all info of an answer to a given problem
#
class ProblemAnswer:
        def __init__(self, answer, is_correct, is_hint):
                self.answer = answer
                self.count = 0
                self.is_correct = is_correct
                self.is_hint = is_hint
                self.is_common = False
                self.next_question_correct = 0
                self.next_question_incorrect = 0
                self.prediction = 0

#
# class that holds the related info to a prediction
#
class PredictionEntries:
        def __init__(self, order_id, assignment_id, user_id, problem_id, problem_used, answer_text,
answer_type, correctness, wrong_answer_correct_rate):
                self.order_id = order_id
                self.assignment_id = assignment_id
                self.user_id = user_id
                self.problem_id = problem_id
                self.problem_used = problem_used
                self.correctness = correctness
                self.answer_text = answer_text
                self.answer_type = answer_type
                self.wrong_answer_correct_rate = wrong_answer_correct_rate

#
# class that holds the value of a prediction
#
class Prediction:
        def __init__(self, prediction, count):
                self.prediction = prediction
                self.count = count

#
# function that selects only problems with interesting common wrong answers
#

def getProblemSet(problem_dic, min_difference, min_num_response):
        selected_problem_set = Set()
```

```
	for problem_id in		problem_dic:
		current_problem = problem_dic[problem_id]
		wrong_answer_correct_rate = current_problem.wrong_answer_correct_rate

		for answer in current_problem.answers:
			if answer != "CORRECT" and answer != "HINT" and answer != "UNCOMMON":
				if current_problem.answers[answer].prediction > wrong_answer_correct_rate
+ min_difference or current_problem.answers[answer].prediction < wrong_answer_correct_rate - min_difference:
					if current_problem.answers[answer].count > min_num_response:
						selected_problem_set.add(problem_id)
	return selected_problem_set


#
# function that makes predictions on the selected problem set with interesting common wrong asnwers
#

def calculateResults(data_frame, p_selected_problem_set, min_difference, min_num_response):
	truth_array = []
	prediction_array = []
	badly_grained_predictions = []
	count_array = []

	for index, row in enumerate(data_frame[1:]):
		order_id = row[iorder_id]
		problem_id = row[iproblem_id]
		answer_text = row[ianswer_text].strip(" ")
		hint_count = row[ihint_count]
		correct = row[icorrect]
		user_id = row[iuser_id]
		assignment_id = row[iassignment_id]
		skill_id = row[iskill_id]
		answer_type = "UNCOMMON"
		entry_count = 0

		if answer_text == "" and hint_count > 0:
			answer = "HINT"
		elif correct == "1":
			answer = "CORRECT"
		else:
			answer = answer_text

		#
		# perform the prediction on the test set
		#
		if int(user_id) % MOD_NUM == MOD_GROUP:
			#
			# locate th next problem first
			#

			prediction = -1
			badly_grained = -1
```

```
                    if problem_id in p_selected_problem_set:
                        current_problem = problem_dic[problem_id]
                        answers = current_problem.answers
                        if answer in answers:
                            if answer == "CORRECT":
                                #
                                # generate the prediction for the correct problem log entries
                                #
                                prediction = answers[answer].prediction
                                badly_grained = answers[answer].prediction
                                entry_count = answers[answer].count;
                                if entry_count < 6:
                                    prediction = correct_correct_percentage
                                answer_type = "CORRECT"
                            elif answer == "HINT":
                                #
                                # generate prediction for the hint problem log entries
                                #
                                prediction = answers[answer].prediction
                                badly_grained = answers[answer].prediction
                                entry_count = answers[answer].count;
                                if entry_count < 6:
                                    prediction = hint_correct_percentage
                                answer_type = "HINT"
                            elif answers[answer].is_common:
                                #
                                # if the answer is a common wrong answer
                                #
                                prediction = answers[answer].prediction

                                badly_grained =
current_problem.wrong_answer_correct_rate
                                entry_count = answers[answer].count;
                                if prediction > badly_grained + min_difference or prediction <
badly_grained - min_difference:
                                    if entry_count > min_num_response:
                                        answer_type = "COMMON"
                                else:
                                    answer_type = "UNCOMMON"
                            elif "UNCOMMON" in answers:
                                #
                                # generate the prediction for the uncommon problem log
entries
                                #
                                prediction = answers["UNCOMMON"].prediction
                                badly_grained =
current_problem.wrong_answer_correct_rate
                                entry_count = answers["UNCOMMON"].count;
                                if entry_count < 6:
                                    prediction = uncommon_correct_percentage
                                answer_type = "UNCOMMON"

                        if index + 2 < len(data_frame):
```

```
                                        next_row = data_frame[index + 2]
                                        next_order_id = next_row[iorder_id]
                                        next_skill_id = next_row[iskill_id]
                                        next_user_id = next_row[iuser_id]
                                        next_assignment_id = next_row[iassignment_id]
                                        next_problem_id = next_row[iproblem_id]
                                        next_answer_text = next_row[ianswer_text]
                                        next_correctness = next_row[icorrect]

                                        if next_user_id == user_id and next_assignment_id == assignment_id
and answer_type == "COMMON":
                                                truth_array.append(PredictionEntries(next_order_id,
next_skill_id, next_user_id, next_problem_id, problem_id, next_answer_text, answer_type, next_correctness,
current_problem.wrong_answer_correct_rate))
                                                prediction_array.append(prediction)
                                                count_array.append(entry_count)
                                                badly_grained_predictions.append(badly_grained)

    #
    # calculates the student level effectsize
    #

            # effect_size = calculateEffectSize(truth_array, prediction_array, badly_grained_predictions)
            # return effect_size

            measurePerformance(truth_array, prediction_array, badly_grained_predictions, count_array)

            #
            # calculate the averaged common wrong answer count
            #
            # getProblemResponseCount(p_selected_problem_set, problem_dic)
            # return len(truth_array)

#
# function that measures the RMSE and the R^2 values given the predictions
#
# @param prediction_array array of ICWAs predictions
# @param badly_grained_predictions array of Control Model predictions
#
def measurePerformance(truth_array, prediction_array, badly_grained_predictions, count_array):
            sum_of_square = 0
            sum_of_truth_average_square = 0
            average_truth_value = 0
            bad_sum_of_square = 0
            entries = 0
            correct_entries = 0

            prediction_outfile = open("predictions.csv", "w")
            badly_grained_outfile = open("badly_grained_predictions.csv", "w")

            prediction_outfile.write("order_id,skill_id,user_id,problem_id,problem_used,type,wrong_answer_rate,co
rrectness,prediction,badly_grained_prediction,count\n")
```

```
	badly_grained_outfile.write("order_id,skill_id,user_id,problem_id,correctness,prediction,count\n")

	#
	# calculate the average of the truths in the truth array for R square calculation
	#
	for index in range(0, len(truth_array)):
		truth = float(truth_array[index].correctness)
		average_truth_value += truth

	average_truth_value = average_truth_value / len(truth_array)

	for index in range(0, len(truth_array)):
		order_id = truth_array[index].order_id
		truth = float(truth_array[index].correctness)
		type = truth_array[index].answer_type
		assignment_id = truth_array[index].assignment_id
		user_id = truth_array[index].user_id
		problem_id = truth_array[index].problem_id
		problem_used = truth_array[index].problem_used
		wrong_answer_correct_rate = truth_array[index].wrong_answer_correct_rate
		answer_text = truth_array[index].answer_text

		prediction = prediction_array[index]
		badly_grained_prediction = badly_grained_predictions[index]
		count = count_array[index]

		if prediction != -1 and badly_grained_prediction != -1:
			sum_of_square += (prediction - truth) * (prediction - truth)
			sum_of_truth_average_square += (truth - average_truth_value) * (truth -
average_truth_value)
			bad_sum_of_square += (badly_grained_prediction - truth) * (badly_grained_prediction -
truth)
			entries += 1
			if prediction == truth:
				correct_entries += 1

		if prediction != -1 and badly_grained_prediction != -1 :
			prediction_outfile.write(order_id + "," + assignment_id + "," + user_id + "," + problem_id
+ "," + problem_used + ","+ type + "," +  str(wrong_answer_correct_rate) + "," + str(truth) + "," + str(prediction) +
"," + str(badly_grained_prediction) + "," + str(count) + "," + "\n")
			badly_grained_outfile.write(order_id + "," + assignment_id + "," + user_id + "," +
problem_id + "," + str(truth) + "," + str(badly_grained_prediction) + "," + str(count) + "," + "\n")

	print(1 - sum_of_square / sum_of_truth_average_square)
	print(1 - bad_sum_of_square / sum_of_truth_average_square)

	if entries != 0:
		sum_of_square /= entries
		bad_sum_of_square /= entries

	print(math.sqrt(sum_of_square))
	print(math.sqrt(bad_sum_of_square))
	print(entries)
```

```
	prediction_outfile.close()
	badly_grained_outfile.close()

#
# function to get the count of responses for each problem
# @param selected_problem_set the problems that contain interesting common wrong answers
# @param problem_dic the problem dictionary that holds every single problem in the training set
#
def getProblemResponseCount(selected_problem_set, problem_dic):
	common_wrong_answer_problem_file = open("cwasp.csv", "w")
	common_wrong_answer_problem_file.write("problem_id,count\n")

	for problem_id in selected_problem_set:
		total_count = 0
		problem = problem_dic[problem_id]
		total_count = total_count + problem.common_next_incorrect
		total_count = total_count + problem.uncommon_next_incorrect
		common_wrong_answer_problem_file.write(problem_id + "," + str(total_count) + "\n")


	for problem_id in problem_dic:
		total_count = 0
		problem = problem_dic[problem_id]
		total_count = total_count + problem.common_next_incorrect
		total_count = total_count + problem.uncommon_next_incorrect
		if total_count > 100:
			common_wrong_answer_problem_file.write(problem_id + "\n")

	common_wrong_answer_problem_file.close()

#
# function to calculate the effect size of the prediction array compared to the badly grained predictions on a
student level
# @param truth_array the array that contains the actual correctness of the problems
# @param prediction_array the array containing the CWAs predictions
# @param badly_grained_predictions the array containing the predictions from the control model
#
def calculateEffectSize(c_truth_array, c_prediction_array, c_badly_grained_predictions):
	user_prediction_average = {}
	user_prediction_sum = {}
	user_prediction_count = {}

	user_badly_grained_average = {}
	user_badly_grained_sum = {}
	user_badly_grained_count = {}

	for index in range(0, len(c_truth_array)):
		truth = float(c_truth_array[index].correctness)
		user_id = c_truth_array[index].user_id
		prediction = c_prediction_array[index]
		badly_grained_prediction = c_badly_grained_predictions[index]
```

```
                if user_id not in user_prediction_sum:
                        user_prediction_sum[user_id] = prediction
                        user_prediction_count[user_id] = 1
                else:
                        user_prediction_sum[user_id] += prediction
                        user_prediction_count[user_id] += 1

                if user_id not in user_badly_grained_sum:
                        user_badly_grained_sum[user_id] = badly_grained_prediction
                        user_badly_grained_count[user_id] = 1
                else:
                        user_badly_grained_sum[user_id] += badly_grained_prediction
                        user_badly_grained_count[user_id] += 1

        for user_id in user_prediction_sum:
                user_prediction_average[user_id] = user_prediction_sum[user_id] /
user_prediction_count[user_id]

        for user_id in user_prediction_sum:
                user_badly_grained_average[user_id] = user_badly_grained_sum[user_id] /
user_badly_grained_count[user_id]

        average_badly_grained = 0

        for user_id in user_badly_grained_average:
                average_badly_grained += user_badly_grained_average[user_id]

        average_badly_grained /= len(user_badly_grained_average)

        average_prediction = 0

        for user_id in user_prediction_average:
                average_prediction += user_prediction_average[user_id]

        average_prediction /= len(user_prediction_average)

        sum_of_square = 0

        for user_id in user_badly_grained_average:
                sum_of_square += (user_badly_grained_average[user_id] - average_badly_grained) *
(user_badly_grained_average[user_id] - average_badly_grained)

        sum_of_square /= len(user_badly_grained_average)

        stdev = math.sqrt(sum_of_square)

        effect_size = (average_prediction - average_badly_grained) / stdev

        return effect_size

#
# ================================================================================
# Main script
```

```
# ==============================================================================
#


#
# read the data file into memory
#
infile_path = sys.argv[1]
csv.field_size_limit(13107200)
data_file = open(infile_path, "r")
data = csv.reader(data_file, delimiter = ",")
data_frame = []

for row in data:
	data_frame.append(row)

#
# acquire all wrong answers and organize them into a map from the dataset
#

problem_dic = {}

for row in data_frame[1:]:
	problem_id = row[iproblem_id]
	sequence_id = row[isequence_id]
	answer_text = row[ianswer_text].strip(" ")
	hint_count = row[ihint_count]
	correct = row[icorrect]
	user_id = row[iuser_id]
	is_correct = False
	is_hint = False

	#
	# begin to gather common wrong answer info on the training set
	#
	if int(user_id) % MOD_NUM != MOD_GROUP:
		if problem_id not in problem_dic:
			problem_dic[problem_id] = Problem(problem_id, sequence_id)

		current_problem = problem_dic[problem_id]

		if answer_text == "" and hint_count > 0:
			answer_text = "HINT"
			is_hint = True
		elif correct == "1":
			answer_text = "CORRECT"
			is_correct = True

		if answer_text not in current_problem.answers:
			current_problem.answers[answer_text] = ProblemAnswer(answer_text, is_correct,
is_hint)

		current_problem.answers[answer_text].count += 1
```

```
#
# print the results to the output for inspection
#

outfile = open("model.csv", "w")
cwa_outfile = open("cwas.csv", "w")
matrix_outfile = open("matrix.csv", "w")


for problem_id in problem_dic:
	current_problem = problem_dic[problem_id]
	for answer in current_problem.answers:
		if answer != "CORRECT" and answer != "HINT":
			current_problem.total_entries += current_problem.answers[answer].count


for problem_id in problem_dic:
	current_problem = problem_dic[problem_id]
	for answer in current_problem.answers:
		if answer != "HINT" and answer != "CORRECT" and current_problem.answers[answer].count >
current_problem.total_entries * 0.1:
			current_problem.answers[answer].is_common = True

#
# at this point I have the dictionary to all common wrong answers, now, for predictions
#

for index, row in enumerate(data_frame[1:]):
	problem_id = row[iproblem_id]
	answer_text = row[ianswer_text].strip(" ")
	hint_count = row[ihint_count]
	correct = row[icorrect]
	user_id = row[iuser_id]
	assignment_id = row[iassignment_id]
	skill_id = row[iskill_id]

	if answer_text == "" and hint_count > 0:
		answer = "HINT"
	elif correct == "1":
		answer = "CORRECT"
	else:
		answer = answer_text

	#
	# build the prediction table using the training data
	#
	if int(user_id) % MOD_NUM != MOD_GROUP:
		if index + 2 < len(data_frame):
			#
			# current row is on experiment_data_frame[index + 1]
			#
			next_row = data_frame[index + 2]
```

```
                        #
                        # confirm that the next row is the student's next problem log in the same assignment
                        #
                        next_skill_id = next_row[iskill_id]
                        next_assignment_id = next_row[iassignment_id]
                        next_user_id = next_row[iuser_id]
                        next_answer_text = next_row[ianswer_text]
                        next_correct = str(next_row[icorrect]).strip(" ")

                        if next_assignment_id == assignment_id and next_user_id == user_id:
                                if next_correct == "1":
                                        problem_dic[problem_id].answers[answer].next_question_correct +=
1
                                else:
                                        problem_dic[problem_id].answers[answer].next_question_incorrect
+= 1

#
# get the calculation for everything
#
total_correct_correct = 0
total_correct_incorrect = 0
total_hint_correct = 0
total_hint_incorrect = 0
total_common_correct = 0
total_common_incorrect = 0
total_uncommon_correct = 0
total_uncommon_incorrect = 0

outfile.write("problem_id,answer_text,correctness,count\n");
cwa_outfile.write("sequence_id,problem_id,answer_text,correctness,count\n");

for problem_id in problem_dic:
        current_problem = problem_dic[problem_id]
        uncommon_next_correct = 0
        uncommon_next_incorrect = 0

        for answer in current_problem.answers:
                next_correct = current_problem.answers[answer].next_question_correct
                next_incorrect = current_problem.answers[answer].next_question_incorrect
                total_entries = next_correct + next_incorrect
                if answer == "CORRECT":
                        if total_entries != 0:
                                total_correct_correct += next_correct
                                total_correct_incorrect += next_incorrect
                                current_problem.answers[answer].prediction = float(next_correct) /
float(total_entries)
                                outfile.write(problem_id + ",CORRECT," + str(float(next_correct) /
float(total_entries)) + "," + str(total_entries) + "\n")
                elif answer == "HINT":
                        if total_entries != 0:
                                total_hint_correct += next_correct
                                total_hint_incorrect += next_incorrect
```

```
                                        current_problem.answers[answer].prediction = float(next_correct) /
float(total_entries)
                                        outfile.write(problem_id + ",HINT," + str(float(next_correct) /
float(total_entries)) + "," + str(total_entries)  + "\n")
                        elif current_problem.answers[answer].is_common:
                                if total_entries != 0:
                                        total_common_correct += next_correct
                                        total_common_incorrect += next_incorrect
                                        current_problem.common_next_correct += next_correct
                                        current_problem.common_next_incorrect += next_incorrect
                                        current_problem.answers[answer].prediction = float(next_correct) /
float(total_entries)
                                        outfile.write(problem_id + ",\"" + str(answer) + "\"," + str(float(next_correct) /
float(total_entries)) + "," + str(total_entries)  + "\n")

                                        #
                                        # record the common wrong answers inside an outfile
                                        #
                                        cwa_outfile.write(current_problem.sequence_id + "," + problem_id + ",\"" +
str(answer) + "\"," + str(float(next_correct) / float(total_entries)) + "," + str(total_entries)  + "\n")

                        else:
                                uncommon_next_correct += next_correct
                                uncommon_next_incorrect += next_incorrect
                                total_uncommon_correct += next_correct
                                total_uncommon_incorrect += next_incorrect

                if uncommon_next_correct + uncommon_next_incorrect != 0:
                        if "UNCOMMON" not in current_problem.answers:
                                current_problem.answers["UNCOMMON"] = ProblemAnswer("UNCOMMON", False,
False)
                                current_problem.answers["UNCOMMON"].count = uncommon_next_correct +
uncommon_next_incorrect
                        current_problem.answers["UNCOMMON"].prediction = float(uncommon_next_correct) /
float(uncommon_next_incorrect + uncommon_next_correct)
                        current_problem.uncommon_next_correct = uncommon_next_correct
                        current_problem.uncommon_next_incorrect = uncommon_next_incorrect
                        outfile.write(problem_id + ",UNCOMMON," + str(float(uncommon_next_correct) /
float(uncommon_next_incorrect + uncommon_next_correct)) + "," + str(uncommon_next_correct +
uncommon_next_incorrect) + "\n")

                if uncommon_next_correct + uncommon_next_incorrect + current_problem.common_next_correct +
current_problem.common_next_incorrect != 0 :
                        current_problem.wrong_answer_correct_rate = float(uncommon_next_correct +
current_problem.common_next_correct) / float(uncommon_next_correct + uncommon_next_incorrect +
current_problem.common_next_correct + current_problem.common_next_incorrect)

#
# print out overall correctness information
#
uncommon_correct_percentage = float(total_uncommon_correct) / float(total_uncommon_correct +
total_uncommon_incorrect)
common_correct_percentage = float(total_common_correct) / float(total_common_correct +
```

```
total_common_incorrect)
hint_correct_percentage = float(total_hint_correct) / float(total_hint_correct + total_hint_incorrect)
correct_correct_percentage = float(total_correct_correct) / float(total_correct_correct + total_correct_incorrect)
print("total uncommon correct: " + str(total_uncommon_correct))
print("total uncommon: " + str(total_uncommon_correct + total_uncommon_incorrect))
print("total common correct: " + str(total_common_correct))
print("total common: " + str(total_common_correct + total_common_incorrect))
print("total correct correct:" + str(total_correct_correct))
print("total correct: " + str(total_correct_correct + total_correct_incorrect))
print("total hint correct:" + str(total_hint_correct))
print("total hint: " + str(total_hint_correct + total_hint_incorrect))
print("hint correct percentage: " + str(hint_correct_percentage))
print("uncommon correct percentage: " + str(uncommon_correct_percentage))
print("common correct percentage: " + str(common_correct_percentage))
print("correct correct percentage: " + str(correct_correct_percentage))


#
# get the problem set that contains default ICWAs
#
new_selected_problem_set = getProblemSet(problem_dic, MIN_DIFFERENCE, MIN_RESPONSES)
#
# perform calculations on this problem set
#
calculateResults(data_frame, new_selected_problem_set, MIN_DIFFERENCE, MIN_RESPONSES)

outfile.close()
cwa_outfile.close()
data_file.close()
matrix_outfile.close()
```

SQL Code for extracting the data set from the Assistments database:

```
drop table if exists temp_data cascade;


create table temp_data as
select pl.id as order_id, pl.assignment_id, pl.user_id as user_id, pl.assistment_id as assistment_id, pl.problem_id as
problem_id, pl.original as original, pl.correct, pl.attempt_count, pl.first_response_time as ms_first_response,
pl.tutor_mode, pt.name as answer_type, ca.sequence_id, ca.student_class_id, ca.position, s2.type,
case when si.copied_from is null then ca.sequence_id else si.copied_from end as base_sequence_id, ptsa.skill_id,
sk.name as skill_name, tc.teacher_id, ur.location_id as school_id, pl.hint_count, pl.start_time, pl.overlap_time,
CASE WHEN a.parent_id is null THEN pl.assistment_id ELSE a.parent_id END as template_id, pl.answer_id,
pl.answer_text, pl.first_action, pl.bottom_hint
from problem_logs pl
left outer join problems p on pl.problem_id = p.id
left outer join problem_types pt on p.problem_type_id = pt.id
left outer join class_assignments ca on pl.assignment_id =  ca.id
left outer join sequences s on ca.sequence_id = s.id
left outer join sections s2 on s.head_section_id = s2.id
left outer join sequence_infos si on ca.sequence_id = si.sequence_id
left outer join problem_to_skill_associations ptsa on p.id = ptsa.problem_id
left outer join skills sk
on ptsa.skill_id = sk.id
left outer join teacher_classes tc on tc.student_class_id = ca.student_class_id
left outer join student_classes sc on sc.id = ca.student_class_id
left outer join user_roles ur on ur.user_id = pl.user_id
left outer join assistment_infos a on pl.assistment_id = a.assistment_id
where
ca.student_class_id is not null and
ur.type = 'Student' and ur.location_type = 'School'
and pl.correct is not null
and s2.type = 'MasterySection'
and ca.assignment_type_id not in (6,7)
and original = 1
and pt.id = 4
order by sk.id, pl.user_id, pl.id;


select * from temp_data where first_action = 0 or first_action = 1 order by user_id, assignment_id, order_id;
```