Improving Student Question Classification

Cecily Heiner and Joseph L. Zachary
{cecily,zachary}@cs.utah.edu
School of Computing, University of Utah

Abstract. Students in introductory programming classes often articulate their questions and information needs incompletely. Consequently, the automatic classification of student questions to provide automated tutorial responses is a challenging problem. This paper analyzes 411 questions from an introductory Java programming course by reducing the natural language of the questions to a vector space, and then utilizing cosine similarity to identify similar previous questions. We report classification accuracies between 23% and 55%, obtaining substantial improvements by exploiting domain knowledge (compiler error messages) and educational context (assignment name). Our mean reciprocal rank scores are comparable to and arguably better than most scores reported in a major information retrieval competition, even though our dataset consists of questions asked by students that are difficult to classify. Our results are especially timely and relevant for online courses where students are completing the same set of assignments asynchronously and access to staff is limited.

1 Introduction

Students often ask their questions and express their information needs incompletely. Consequently, the automatic classification of student questions, with the goal of ultimately providing automated tutorial responses, is a challenging problem. For example, the following are information requests from novice programming students:

- "How do i [*sic*] return the file extension only?"
- "I need help extracting a file extension from a filename."

Although phrased differently, both sentences indicate the same need, namely help with the file extension extraction problem; therefore, they should be classified the same way.

This paper classifies student questions by matching them to previous questions with similar meanings but different phrasings. We deployed a software system in an introductory computer science course for approximately one semester to collect ecologically valid data. The system mediated and logged help requests between students and teaching assistants(TAs), capturing both the students' natural language and the associated Java files. The goal of this phase of the research is to quantitatively compare various approaches of classifying the questions that novice programming students ask. The ultimate goal is to be able to provide automated answers to free form student questions by recycling answers to similar previous questions.

2 Prior Work

The AutoTutor project has researched a number of different analytical approaches for processing student language in response to tutorial prompts. They demonstrated that Latent Semantic Analysis(LSA)[8] and cosine similarity with natural language were viable approaches to selecting text for intelligent tutoring dialog with human raters as the

gold standard[13, 14]. The PedaBot project followed a similar line of research with a few fundamental differences. First, the PedaBot project matched student discussions to similar previous student discussion[7]. Because students are notoriously bad at articulating their discussion points, matching student input to student input is a more difficult problem than matching student input to expert-provided input. Second, although the PedaBot approach did not require expert-provided answers, it did require a list of expert-provided technical terms. The PedaBot project avoided generating these manually by automatically extracting them from a textbook or other authoritative, expert provided resource[7]. Like the AutoTutor group, the PedaBot group examined various techniques for calculating similarity of the discussions in the system, with the focus on LSA and cosine similarity[7].

Together, these groups have demonstrated convincingly that LSA and cosine similarity are a promising direction for processing tutorial dialogue, but the general approach still has a number of serious weaknesses. First, the research results are not as compelling as they could be. The AutoTutor group reports correlations with $r < 0.5$[13], and the PedaBot group reports finding discussions of "moderate relevance" or discussions that rank three on a four point Likert scale[7]. Second, the approaches outlined require significant expert-authored resources, either in the form of a list of ideal answers in the case of AutoTutor or in the form of a list of technical terms for PedaBot, and matching these technical terms is critical to both approaches. However, students (especially novice programming students), often do not use technical vocabulary in articulating questions. Third, the approaches seem to rely on students being unrealistically verbose in their interactions with the system. In the AutoTutor dataset, the average length of student responses was 18 words[13], and in our dataset, after stop words are removed, the median length of a student question is six terms. Literature in the information retrieval community has shown that longer queries are often more effective and robust[2], and LSA is most effective with between 300 and 500 terms in the final matrix[3, 13].

By contrast, work in the information retrieval community has generally focused on the query or perhaps a question as the articulation of a user's information needs. A typical web query is between two and three words in length (e.g. [2]) which is quite a bit shorter than a discussion. Although a typical factoid question is longer than two or three words, it is also quite short compared to a discussion. Providing automated answers to factoid questions extracted from community question answering services has been extensively studied as part of the Question Answering Track at Text Retrieval Conference (TREC) (e.g. [12]). Later versions of the TREC competition utilized more difficult datasets and more difficult tasks. Consequently, the scores in later years of the competition were often lower (e.g. [4]), and comparing TREC results across years is like comparing apples and oranges. The relatively low TREC competition scores suggest that answering questions is a difficult task, even without the extra complications from student generated data.

One of the best systems submitted to TREC-9, LCC-SMU, specifically mentions exploiting a technique called "answer caching" to provide answers to some questions that utilize different wordings to express the same information need[9]. Answer caching is a technique that matches an incoming question to a similar previous question (or group of questions with the same answer) in order to recycle an answer. The original paper on

answer caching reports a 1-3% improvement on a dataset with well-formed, grammatical, well-spelled questions. This paper reports a similar result on a more difficult dataset.

Classifying the questions that students ask is a more challenging task than automatically answering factoid questions for several reasons. First, most questions that students ask tend to be about assignments and exams(e.g. [6]) instead of factoids. Factoid questions can usually be answered with a word or a phrase, while questions about assignments and exams generally require answers with one or more sentences. Thus, the space of correct answers for questions that students ask is much larger than the space of correct answers to a factoid question, and the larger space makes question classification more difficult. Second, questions asked by students in a class exist in an extensive educational context, so the question "How do I draw a pyramid?" has a very different meaning in an introductory programming class than it would in an introductory art class. Third, questions asked in class are often of a more subjective nature, such as coding style, but factoid questions are often of a more objective nature. Fourth, the questions students ask tend to be ungrammatical and contain typos and spelling errors.

3 Data

Questions asked in Introduction to Computer Science 1 (CS1410) at the University of Utah form the dataset for this paper. Most students in Computer Science 1 are age 18-22. Computer Science 1 is the first required computer science course for computer science majors, with a strong emphasis on the Java programming language. The course has long hours for novice programmers and typically high dropout, fail, and withdrawal rates. The majority of students who take Computer Science 1 hope to major in computer science or a related field, but they must pass that class along with three others with sufficiently high grades to attain official status as a computer science major. Although approximately 233 students were active in the course during the study period, only 63 of them asked questions while using the study's logging software during the study period.

The goal of this research is to classify the questions that students ask by automatically identifying similar previous questions. To facilitate analysis of student questions, a proprietary software system logged the questions that the students asked and the accompanying source code during the study period. The long term goal is to complete the analysis for a question in real time and exploit it for an instant tutorial intervention. In the interim, when a student asks a question, the system logs the student's question and source code and passes it to a teaching assistant (TA) who can answer the question in person or remotely. The TA then tags the question to indicate an answer category.

We tagged all of the data by associating all questions that could be answered with the same response to the same, unique tag. Then an undergraduate TA tagged approximately 15% of the data, assigning tags from a set devised for that assignment. The TA did not recode the other 85% of the data, but because the inter-rater reliability for the questions we sampled was better than 95%, we included all of the data in the final dataset. This left a dataset of 411 questions from 13 different assignments covering a total of 136 answer categories or information needs. Of the 411 questions, 275 of the questions (136 subtracted from 411) were repetitive in nature, and had a similar previous question. That

means that 66% of the questions were repetitive. Excluding stop words, length of student questions ranged from 0 to 93 terms, with a median of six words and a mode of four words. Approximately 2% of the questions had no terms after stop words were excluded. More than 90% of the questions had 16 terms or fewer.

4 Similarity Scoring

We follow typical practice for processing the natural language in the questions that the students asked. The sentences were tokenized based on spaces and other special characters. Stop words (e.g. "me", "you", and "the") were excluded. Then, a Porter stemmer[10] removed the word endings leaving just the word stem (e.g. the word "extension" became "extens"). The word stems from each question then form a vector. Table 1 shows some sample student questions and the corresponding vectors of stems.

Table 1: Sample Questions, Vector Stems, and Answer Categories

	Natural Language	Vector Stems	Answer Category
Q1	How do i return the file extension only?	return file extens	File extension extraction
Q2	my variable for rectSideOne is suppose to be 1/9, the program is returning a 0 for this calculation. I have no idea why.	Variabl rectsideon suppos 1/9 program return calcul idea	Integer division
Q3	I need help extracting a file extension from a filename.	need help extract file extens filename	File extension extraction
Q4	program is not computing volume correctly	Program comput volum correctly	Integer division
Q5	Im having trouble understanding why (1/9) equals 0.0 instead of 0.111111	trouble understand 1/9 equal	Integer division

The word stems that remained for each question populated a frequency matrix f_{ij}, which gives the number of times word stem j appears in question i. This matrix has 411 rows (one per question) and one column per unique word stem. The questions were compared in the order they were originally asked by the students to all previously asked questions. Specifically, we used cosine similarity (Equation 1) to compare question i against each of questions 1 through i-1, using weights w_{ij} computed from the frequencies f_{ij} with standard term frequency, inverse document frequency (tfidf) weighting. The tfidf weights were recomputed each time the algorithm advanced to the next question, which effectively enlarged the model by one question.

The remainder of the analysis utilizes an online learning framework to identify similar previous questions. Each question is compared to all previous questions, and the previous question with the highest cosine similarity score (as shown in Equation 1) when compared to this question is considered the most similar. If the current question and the most similar question have the same answer category, the system earns a point for accuracy. For example, in Table 1, Q2 would only be compared to Q1, and the system would not earn a point for accuracy. However, Q5 would be compared to Q1, Q2, Q3, and Q4. Of these, Q2 would be the most similar, and since Q2 and Q5 share an answer category, the system would earn one point for accuracy.

$$\frac{\sum_{j=1}^{t} w_{cj} w_{pj}}{\sqrt{\sum_{j=1}^{t} (w_{cj})^2} \sqrt{\sum_{j=1}^{t} (w_{pj})^2}}$$

Equation 1: Cosine Similarity

5 Analysis and Results

5.1 Baseline Cosine Similarity

As shown in Figure 1 and Table 2, we report accuracy scores with three different denominators. In total questions, 411 is always the denominator. In repetitive questions, either 275 or 204 is the denominator depending whether or not the data is disaggregated. Of these, only the repetitive questions bar could theoretically reach 100%. In both cases, the numerator is the number of correct similar questions found(93). As a baseline, cosine similarity is applied to the natural language of the students' questions. With that baseline, the algorithm can classify approximately 35% of the repetitive questions or 23% the total questions. For those questions, an answer to a previous question could theoretically be recycled to answer that question.

Table 2: Classification Accuracy Counts and Percentages

	Aggregated		Disaggregated	
	Total Questions	Repetitive Questions	Total Questions	Repetitive Questions
Baseline	93/411 (23%)	93/275 (35%)	113/411 (27%)	113/204 (55%)
With Error Msgs and Answer Cache	104/411 (25%)	104/275 (39%)	111/411 (27%)	111/204 (54%)

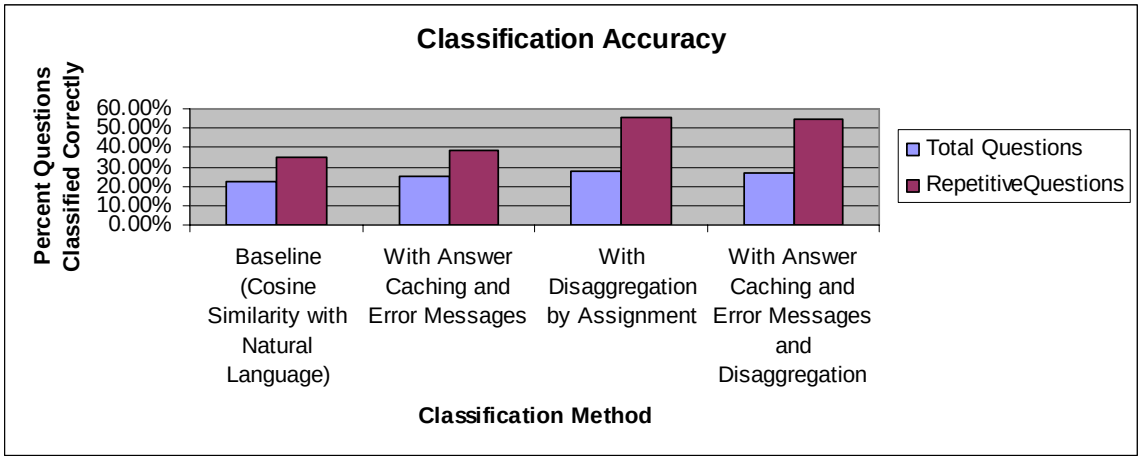

Figure 1 Classification Accuracy

5.2 With Error Messages and Answer Caching

The classification techniques described so far are inherently domain independent. However, the low accuracy of question classification suggests room for substantial improvement. One possible way to improve classification is to leverage some domain specific knowledge, specifically the error messages from the compiler. Since more than 40% of the questions were submitted with code that did not compile, the compiler error messages represent a source of substantial unused data.

A naïve approach to incorporating compiler output would be to simply tokenize the errors and include them just as the natural language was included. The problem is that errors such as missing import and capitalization will appear to be very similar because they contain four similar tokens ("cannot", "find", "symbol", and "class"), and the algorithm will be unable to distinguish between them. To remediate this problem, some of the most common compiler errors and code snapshots are processed by Java code that generates a brief description of the underlying error based on the code snapshot, and then the underlying error is incorporated into the model. For example, the common error message "cannot find symbol- class Scanner" is processed and becomes "missingImport", and the common error message "cannot find symbol –class string" becomes "capitalization".

Previous work has exploited a technique called "answer caching" to provide answers to some questions that utilize different wordings to express the same information need[10]. Answer caching matches an incoming question to a similar previous question in order to recycle an answer. The answer caching technique then leverages the additional language in the similar question to build a more robust language model of that information need. Specifically, answer caching merges the data from vectors that indicate a similar information need to form a single vector. Without answer caching, the five questions in Table 1 are modeled with five vectors. With answer caching, they are represented with two vectors, one for "File extension extraction"(the sum of the vectors for Q1 and Q3) and one for "Integer division"(the sum of the vectors for Q2, Q4, and Q5).

The original paper on answer caching reports a 1-3% improvement on a dataset with well-formed, grammatical, well-spelled questions. Figure 1 demonstrates a similar improvement when incorporating both answer caching and the processed error messages, even though our data consists of student questions with typos and other complications. Interestingly, the processed error messages alone do not improve classification, and answer caching alone produces a minor improvement of less than 1%, but the combination of the techniques improves accuracy by 3% of the total questions. The numerator in for the "With Answer Caching and Error Messages" method is 104, and the denominators are the same as they were in the baseline conditions, 411 for total questions and 275 for repetitive questions.

5.3 Disaggregating by Assignment

For a final improvement in classification accuracy, the data was disaggregated by assignment. For example, assignment1 questions were compared only to other assignment1 questions and assignment5 questions were compared only other assignment5

questions. As shown in Figure 1, this technique improved the numerator to 113 questions classified correctly or 27% of total questions and 55% of repetitive questions

With the data disaggregated by assignment, incorporating answer caching and error messages reduced accuracy slightly(111 questions classified correctly). The lack of sufficient data to model different kinds of compiler errors is probably the cause of a drop in accuracy when answer caching and error messages are incorporated. Because compiler errors are being reduced to a single term, several of them are necessary to boost the compiler error terms to a heavy enough weight to influence the similarity algorithm. Excluding error messages and answer caching returns the classification algorithm to a domain independent state. Compiler error messages are a source of data that are only relevant in the computer science domain. By contrast, natural language and assignment numbers are a data source that is available in virtually every educational domain.

5.4 Discussion

We have shown that the baseline algorithm can be improved by incorporating an answer caching/compiler error extension (an additional 4% of repetitive questions classified.) This improves on the earlier results on answer caching[9], and is especially noteworthy given that we obtained our results on a student-generated corpus.

Most importantly, we have shown that the baseline algorithm can be improved by disaggregating by assignment (an additional 7% of repetitive questions classified.) In fact, this percentage substantially understates the actual improvement that we observed. To facilitate comparison in the bar charts, we use the same denominators throughout. However, when comparing questions only within the same assignment, the number of repetitive questions is actually smaller (204). Using that as the denominator yields a classification accuracy of 55% of repetitive questions.

Classification accuracies of 55% are neither great nor terrible. They are good enough that a desperate student who is working on an assignment at midnight might actually be able to find a useful bit of information when a human TA is not on duty. In such a situation, a bad answer may be better than no answer. However, they are low enough to raise concern that the system may not answer student questions correctly, and worse, the system might lead the student astray. At least two alternatives are possible intermediate steps to deploying this in a real classroom. First, the existing corpus could be leveraged as a starting point for designing common error detectors and appropriate interventions. Second, a human TA could supervise the classification algorithm, and override any incorrect decisions that it makes, until the number of incorrect decisions decreases.

6 Limitations and Future Work

6.1 Classification Schemes for Questions that Novice Programmers Ask

Given a set of categories, classifying questions appears to be relatively straightforward for humans. However, no widely accepted set of categories or taxonomy exists for the questions that novice programmers ask. Previous work has suggested either 42, 88, or

226 different categories for compiler errors [1, 5, 11], and compiler errors only account for half of the questions in the data set presented in this paper. Those papers are simply trying to classify compiler errors based on the compiler error message, not the underlying misconception the student has expressed. Furthermore, a single piece of code may have multiple issues. Ideally, students would request help at the point of a partial impasse, but students appear to frequently wait until they have reached a full impasse before requesting help. The resulting code often has many problems. In this study, such a question would have probably been assigned a label that encompasses a broad range of problems. Work on classification schemes that allow free-response student-input to be assigned multiple, more-fine-grained designations would be applicable for question classification as well as other problems. That research will probably also require work on partial parsing, and other approaches for handling poorly formed student input that cannot be parsed with readily available tools.

6.2 Usability issues

A number of usability issues on both the teacher and the student side must be resolved before an automatic question answering system can be deployed in a classroom setting. On the teacher side, training may be necessary to classify student questions correctly. Once automated interventions are added, the teacher will need to determine if the student still needs human help because the system classified the question incorrectly or because the automated intervention was ineffective. On the student side, studies should investigate whether or not a drop-down menu of frequently asked questions can help students articulate their questions, and whether or not students accept automated answers to their questions, especially if they know that a human TA is on duty and available.

6.3 Model of time spent

The data collected for this study could be reanalyzed to build a model of how long it takes to answer a particular question taking into account factors such as the student asking the question, the question that was asked, and the teacher answering the question. Such a model could help answer questions about which factors are most important in predicting the amount of time it will take to answer a question. Such a model may also allow the system to automatically perform triage, determining which questions are the quickest and most urgent to answer and suggesting that a teacher answer those first, thus reducing total student wait time. However, some students who are accustomed to first in first out service might complain that such an approach is unfair.

6.4 Improving the feature set and data set

A larger dataset may support stronger claims and possibly allow interesting disaggregations in different dimensions. For example, with more data, the combination of disaggregating by assignment and including error messages and answer caching may be more accurate than disaggregating by assignment alone. The feature set could be better. For example, compound words are not well analyzed, so a question containing the words “monthly” and “payment” may not have any terms in common with a question containing the word “monthlyPayment” even though they are clearly semantically similar.

Additionally, many compiler errors and features that could be extracted from student source code, such as extra semicolons before the body of a loop, are ignored.

7 Contributions and Conclusions

We show that cosine similarity is a non-trivial baseline for experiments to improve accuracy in question classification. We discuss previous results showing that answer caching can improve accuracy by 1-3% and extend previous work on answer caching by achieving similar improvement on a more difficult dataset and demonstrating that it is a valid approach for tutorial dialogue by utilizing an ecologically valid dataset. We demonstrate that additional improvements in accuracy are possible by exploiting other sources of data beyond the natural language of student questions, and we demonstrate additional modest gains in accuracy using some compiler errors. These techniques produce a 4-7% improvement in question classification accuracy, bringing total question classification accuracy to 28% of all questions or 42% of repetitive questions, or 56% of repetitive questions when disaggregated by assignment.

Our classification methods work over half the time for student-generated questions, assuming that the questions can be separated by assignment. Thus, our methods would work particularly well in a course in which the same assignments are used over and over, and our long-term goal of using a classification-based approach to automatically answer questions would be especially valuable in a course in which students have low access to course staff. These two conditions are typical of online classes, which represent a fast growing segment of courses in higher education.

Acknowledgements

This work was partially supported by the National Science Foundation, under REESE Grant No. DRL-0735264. Thank you to the students and instructors who used the data collection system. Peter Hastings provided an excellent stopword list. Thank you to reviewers who have provided useful feedback to improve this research.

References

[1] Ahmadzadeh, M., Elliman, D., Higgins, C., An analysis of patterns of debugging among novice computer science students, in Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education. 2005, ACM: Caparica, Portugal.

[2] Belkin, N. J., Kelly, D., Kim, G., Kim, J. Y., Lee, H. J., Muresan, G., Tang, M. C., Yuan, X. J., Cool, C., Query length in interactive information retrieval, in Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. 2003, ACM: Toronto, Canada.

[3] Bradford, R. B., An empirical study of required dimensionality for large-scale latent semantic indexing applications, in Proceeding of the 17th ACM conference on Information and knowledge management. 2008, ACM: Napa Valley, California, USA.

[4] Dang, H. T., Lin, J., Kelly, D. Overview of the TREC 2006 Question Answering Track. *Text REtrieval Conference*, 2006.

[5] Jadud, M. C., A First Look at Novice Compilation Behaviour Using BlueJ. Computer Science Education, 2005. 15(1): p. 25 - 40.

[6] Kim, J., Shaw, E., Chern, G., Herbert, R. Novel tools for assessing student discussions: Modeling threads and participant roles using speech act and course topic analysis. *Artificial Intelligence in Education*, 2007. IOS Press.

[7] Kim, J., Shaw, E., Ravi, S., Tavano, E., Arromratana, A., Sarda, P., Scaffolding On-Line Discussions with Past Discussions: An Analysis and Pilot Study of PedaBot, in Intelligent Tutoring Systems. 2008. p. 343-352.

[8] Landauer, T. K., Foltz, P. W., Laham, D., An Introduction to Latent Semantic Analysis. Discourse Processes, 1998. 25(2/3): p. 259-284.

[9] Pasca, M. A., Harabagiu, S. M., High performance question/answering, in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. 2001, ACM: New Orleans, Louisiana, United States.

[10] Porter, M. F., An algorithm for suffix stripping. Program, 1980. 14: p. 130-137.

[11] Thompson, S. M., An Exploratory Study of Novice Programming Experiences and Errors. 2006, University of Victoria. p. 153.

[12] Voorhees, E. M. Overview of the TREC 2001 Question Answering Track. *Text REtrieval Conference (TREC)*, 2001.

[13] Wiemer-Hastings, P., Wiemer-Hastings, K., Graesser, A. How Latent is Latent Semantic Analysis? *Proceedings of the 16th International Joint Congress on Artificial Intelligence*, 1999. Morgan Kaufmann.

[14] Wiemer-Hastings, P., Wiemer-Hastings, K., Graesser, A. C. Approximate natural language understanding for an intelligent tutor. *12th International Florida Artificial Intelligence Research Conference*, 1999. AAAI Press.