Building Predictive Models for CS Students Help-Seeking Behaviors with Coding Log Data

Zhikai Gao
North Carolina State University
zgao9@ncsu.edu

Collin Lynch
North Carolina State University
cflynch@ncsu.edu

ABSTRACT

Promptly addressing students' help requests in office hours has become a rising challenge for instructors in large CS courses with heavy coding assignments. Therefore, in this research, we propose several different predictive models based on students' coding activity data to predict when students will seek help and what questions they will ask. Additionally, in order to solve the problem of help-abuse, we propose a method to quantify students' coding efforts before and after they seek help. We describe our plan and designs thoroughly in this paper and hope to receive constructive feedback on any aspect of this study. We believe this study could make a huge contribution to improving the learning experience in office hours.

Keywords

Office Hours, Help-seeking, Coding Behavior, CS2, Predictive Models

1. INTRODUCTION

Computer Science has witnessed a significant rise in popularity as a chosen academic discipline. With such high demand, course sizes have also expanded. Consequently, instructors are currently grappling with a substantial amount of students' help requests. Addressing those requests properly and promptly is important, but it is still quite challenging in any large course.

To address those help requests, instructors can host a one-on-one meeting with a student at a scheduled time; this method is called Office Hours. During the meeting, students can directly communicate with the instructor about any questions related to the course. Compared with other help-seeking approaches, like public online post forums[4] or LLM chatbots[16], office hours offer a more personalized, interactive, and direct interaction. Traditionally, office hours are hosted in person; the teacher sits in their office with an open door, and students are welcome to drop by and ask any questions they have. Prior research has shown that this traditional method might be underutilized;[11, 17]; thus, some instructors have started experimenting with online/virtual office hours instead. The teacher will open a Zoom meeting to wait for any help requests from students, and during the interaction, students might share the screen with the teacher, showing the code or errors they received. Since the COVID-19 pandemic, this format has been widely applied, and we have observed how this online format increases the attendance of office hours.[14] However, this online format brought another issue. With the increasing help demand, the office hours queue got longer and longer, and instructors started to notice some students were exploiting office hours as a "hint generator" for their assignments and hoped teachers would tell them what to do when they encountered any minor problem. Since the online format lowers the cost of requesting help, some students just stop trying to resolve problems or debugging errors by themselves and rely on office hours. This excessive usage of help resources, called help-abuse[18], is becoming increasingly common and causing instructors to worry that they might not be able to deliver help to real struggling students in time. While several attempts, like group sessions[13] or split deadlines[2], have been made in the classroom to accommodate the increasing help demand, a more comprehensive approach is still needed.

To better manage the office hours resources and request queue, an office-hours ticketing system called MyDigitalHand (MDH)[20] was developed to facilitate support interactions between students and TAs for large CS courses. MDH has been deployed in CS1 and CS2 courses across three universities. To obtain help during office hours, students virtually raise their hands by filling out a form at the MDH website. Students are then placed in the queue. Requests can only be made when an office hours shift has started, and when the teacher is available to help the next student, they can select a student in the queue and inform them to enter the Zoom meeting or enter their office and then start the interaction. After the interaction ended(resolved), the teacher would go back to MDH to close the request ticket in the queue and be ready for the next request. This system provides a great potential to analyze students' office hours behaviors[15, 6, 7, 8, 9, 21, 22, 10], such as when and why they seek help.

Many researchers used surveys to investigate students' view of office hours usage.[22] Others tried to seek the relationship between students' office hours attendance and their perfor-

Z. Gao and C. Lynch. Building predictive models for cs students help-seeking behaviors with coding log data. In B. Paaßen and C. D. Epp, editors, *Proceedings of the 17th International Conference on Educational Data Mining*, pages 1003–1008, Atlanta, Georgia, USA, July 2024. International Educational Data Mining Society.

https://doi.org/10.5281/zenodo.12730031

mance and found contradictory results.[3, 8] However, most of the above research on analyzing students' office hours behaviors either heavily relied on survey data or only studied office hours in isolation. There is little research trying to combine students' help-seeking behavior and other learning activities. Based on our preliminary research[6, 7], CS students are primarily asking questions about their coding assignments; therefore, we believe the correlation between their coding activity and their help-seeking behaviors exists, but no one has investigated yet. Also, there is a lack of research on building predictive models to assist office hours management. Building such models can help us identify students who need help in real-time and estimate the future queue load to assist instructors in better distributing the office hours resources. Therefore, we aim to answer the following questions in this work:

- RQ1: Can we use students' coding status to predict when they will seek help in office hours?
- RQ2: Can we use students' coding status to predict what questions they will ask when they raise an office hours request?
  - RQ2.1: Can we predict what type of question students will ask?
  - RQ2.2: For an implementation request, can we predict which function/part of the code students are struggling with?
  - RQ2.3: For a Test Failure request, can we predict which failed test students are trying to resolve?
- RQ3: How can we measure the necessity or urgency of students' help requests based on their coding behavior and previous help requests?

1.1 Proposed Contribution

By addressing RQ1 and RQ2, we can gain a better understanding of students' coding behavior before they seek help in office hours. Such understanding could in turn inform us in more detail on when and why students are most likely to request help in office hours, as well as provide useful insight for future research on what features in students' code might indicate struggling. RQ1 focuses on predicting when the request might occur, whereas RQ2 focuses on what specific question the student might want to ask.

By exploring when students should seek help in RQ3, we can identify possible solutions to the help-abuse problem. Instructors can use our model to identify and prioritize office hours requests from students who have tried enough solutions but are still struggling. Moreover, this understanding can help students too; we can automatically encourage students who didn't make enough effort before requesting to try more solutions; or inform students to seek help when they are truly struggling.

2. CURRENT AND PROPOSED WORK

2.1 Data Collection

We have collected the office hours usage and GitHub code commit data for this work, which originates from the 9 offerings(Fall 2017 - Fall 2021) of an undergraduate CS2 Java Programming course at a research-intensive university in US. The course initially offered both online and in-person sections; however, due to the COVID-19 pandemic, all undergraduate course offerings in Fall 2020 and Spring 2021 were moved online a few weeks into the semester. The overall structure of the course, as well as the task deadlines, remained the same with interactions, including office hours, moving to individual web meetings but retaining their overall structure. Although the lecture meetings have returned to in-person since Fall 2021, the office hours still remain fully online. In this study, we will only focus on online office hours since it is the format that will be continuously used in the future. Students in the course complete 12 lab assignments, three guided projects, and two main projects. Additionally, we also plan to collect more data from recent semesters(Spring 2022- Spring 2024).

Our office hours data are collected through MDH, which includes the request time, start time, end time, and students' descriptions of their questions. Table 1 shows the number of requests we collected each semester.

We collected the code commit data through the GitHub repo for each student's assignment. During their programming assignment, students periodically submitted their code solution to GitHub through a commit push, and then their code will be tested by a series of black-box unit tests called teaching staff test cases; the final test report will be given back to them automatically so that they can improve their code solution. In this study, we collect the exact code and their test report for each code commit.

Table 1: Number of office hours requests and students for each semester (F= Fall, S=Spring)

In-person	F17	S18	F18	S19	F19
requests	1146	609	1224	860	1650
students	208	157	259	174	256
Online	S20	F20	S21	F21	
requests	1401	3452	2509	3270	
students	191	303	289	405	

2.2 Study Design

The overall study design and methodology are shown in Figure 1. We proposed five models to build in this study. Model 1-4 are predictive models to answer RQ1 and RQ2 based on the code features extracted from the pre-help commit(the commit before the help request was raised). All four predictive models are classification models at the core. Model 5 is a quantitative model to answer RQ3 by measuring the amount of effort students made before and after they received help from office hours.

2.3 Data Processing

We first combined the office hours data and the code submission (Github commit) data by student and time; then, for each office hours request, we automatically identified one pre-help commit(code submitted before the help request, within one hour) and one post-help commit (code submitted after the help request, within one hour). For each code submission, we plan to extract the following coding features. All coding features are inspired by previous analysis in our team on students' code commits[5].

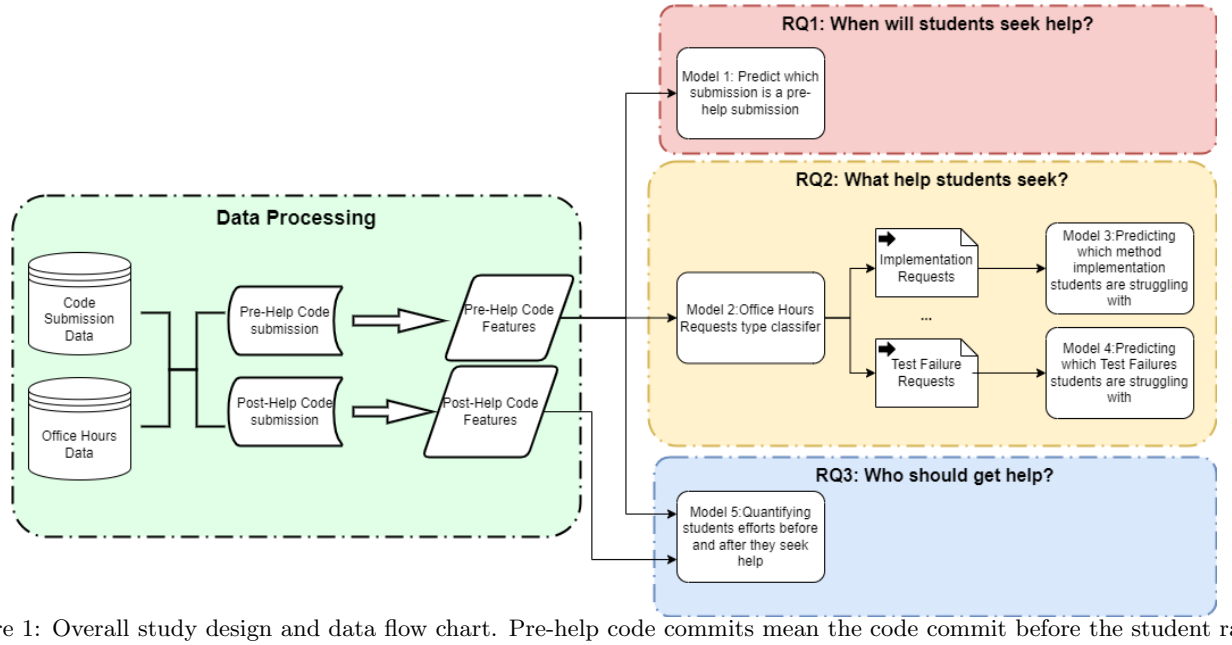

Figure 1: Overall study design and data flow chart. Pre-help code commits mean the code commit before the student raised the office hours request. Post-help code commits mean the commit after an office hours interaction ended.

- The total number of additions, deletions, files changed, and amount of change in each function/file.
- The percentage of additions, deletions, files changed, and amount of change in each function/file.
- The status(Fail or Pass) for each student's self-written tests or teaching staff test.
- Abstract Syntax Tree[23], SANN[12]
- Code embedding, such as code2vec[1]
- The total number and average size of solution functions and solution classes.
- The number of testing methods, assertion density, and statement coverage

After extracting those features from students' code, we will train our predictive models. To train each model, we will first split the train and test dataset with a ratio of 8:2 . We will then apply Random Forest, SVM, GDBT, and LSTM as candidate models and evaluate their performance with accuracy, recall, precision, and F1-score. A detailed discussion of each model and how it can answer the research question are provided in the following subsections.

2.4 Model 1: Predicting When to Seek Help

In RQ1, we would like to predict when students would seek help. Model 1 breaks down this problem to predict which commit will be followed by an office hours request. Specifically, for each commit, if an office hours request exists within one hour and also before the next commit, we call this commit the pre-help commit. This one-hour window is chosen arbitrarily and might be subject to adjustment. With this definition of pre-help commit, our task in Model 1 is essentially to predict whether a commit is a pre-help commit or not. The labeling process for this task can be easily done automatically. Then, we can use the methodology and the coding features we proposed before to build the binary classifier as Model 1.

However, when a student uses office hours, it could vary depending on the individual's preference or habits. With only coding data, our model might not perform well. Therefore, we might also include students' demographic data and previous office hours usage data in the training as well.

Table 2: Preliminary categorization results of office hours requests. Only requests for the main project assignments in F20 and F21 are included.

Category	F20	F21
Implementation	334	222
General Debugging	392	263
TS test failures	127	124
Self-written test failures	30	24
General test failures	103	38
Improving test coverage	21	20
Static analysis notifications	9	7
Others/Unclear	314	102
Total	1330	800

2.5 Model 2: Predicting What Type of Help

In our preliminary research[7], we divided students' office hours requests based on the provided description for the F20 and F21 semesters. Then, two TAs manually labeled each request(Kappa=0.83), and the results are in Table 2. We only focused on requests related to their main projects. Based on the results, we found that students are mostly asking about the implementation, debugging, and test failures of their programming assignments. Additionally, we discovered a significant difference in commit frequency before, during, and after students requested help, depending on the type of assistance they sought. Therefore, we believe it is

Table 3: Pseudo example labeling for RQ 2.2 and RQ 2.3; '-' indicates we can't identify which specific function or unit test students are struggling with and thus need to be removed.

Example Description	Question Type	Label(specific question)
what value should I return in Car.getName() when the name is empty?	Implementation	Car.getName()
How should I calculate the new position of the car?	Implementation	Car.updatePosition()
I don't know how to write updatePosition()	Implementation	Car.updatePosition()
I am confused how to start the project	Implementation	-
testCar.testUpdatePosition() failed	Test failures	testCar.testUpdatePosition()
test failed when car name is empty	Test failures	testCar.testEmptyName()
2 test failures	Test failures	-

crucial to study students' coding behaviors separately based on the type of requests they make.

In order to build a classifier based on the above categorization, we currently still need to address two problems. (1) we only labeled two semesters amount of the data; a large portion of the data stays unlabeled and it would be very time-consuming to do so manually. (2) A high percentage of the data cannot be classified into any meaningful category because of the ambiguous description the students provided. To address these problems, we plan to use semi-supervised learning[19] as the framework to iteratively train the model and classify each unlabeled help request. We first treated every request in the others/unclear category as unlabeled data, combined with new semesters' unlabeled data. At the first iteration, we will train a classifier using labeled data and then use it to predict and pseudo-label the unlabeled data. Then, we select the most confident predictions in the newly labeled dataset and merge them into the labeled dataset; then, we discard the pseudo labels for the remaining unlabeled dataset. We then use both sets of data to re-train the classifier, with this process being repeated over several iterations to get better classifiers. During each iteration, we evaluate the classifier against a separate validation set to ensure that the quality is maintained as we move forward. The training step will stop once the labeled dataset stops increasing, and the model at the last iteration will be our final target model.

2.6 Model 3&4: Predicting Specific Questions

With Model 2, we would successfully categorize the students' requests into different types. In RQ2.2 and RQ2.3, we focus on two major categories: Implementation and Test Failures.

For the Implementation requests, we opt to investigate which function/methods students are asking about. We plan to examine each implementation request manually and label the ones with clear descriptions of which methods they need assistance with. For the rest of the requests that don't clearly specify this information, we will remove them for training this model. Similarly, we will also manually label Test Failure requests with the name of the unit test they reported they failed in the request description. With these two labeled datasets and extracted coding features, we could easily build Model 3 and Model 4.

However, due to the unclear nature of the request description data, we might only get a small labeled dataset for training. To address this problem, we can utilize techniques like data augmentation, data synthesis, cross-validation, and regularization to increase the dataset and prevent overfitting. Additionally, since we have over 40 unit tests and functions for each project, our approach of treating each unit test/function as a single category might be problematic. We might need to perform a clustering algorithm first to merge several functions or unit test categories together.

2.7 Model 5: Quantifying students efforts

In RQ3, our question is to identify students who should receive help based on their coding activity. Our key idea is that if a student makes enough effort to address their problem before they request help, then they are truly struggling and need to be prioritized. Furthermore, prioritizing students who are more likely to put in effort after the interaction before they return for additional assistance can increase the efficiency of office hours. Therefore, in Model 5, our goal is to quantify the students' efforts before they seek help and after they receive help. We will use a series of metrics to represent the amount of effort students made, including commit frequency, time till the last/next improved solution, and coding features listed before. And then, we will calculate the average and median value for each metric as a benchmark for deciding if a student made enough effort; this benchmark might be subject to adjustment if we observe any special distribution of the metric.

3. ADVICE SOUGHT

We are primarily seeking some advice and answers to the following questions:

- Any additional coding features we can extract and utilize in any models we proposed?
- For the question about RQ3 and who should seek help, are there any additional aspects we should think of other than students' coding efforts?
- For Model 3 and Model 4, how can we improve the modeling approach?
- Is there any methodology or design flaw you spotted in this research?
- As a CS educator, do you think our proposed models are useful to you? If so in what way? If not what are your primary concerns about office hours in your class?

In addition to the above questions, we would also be open to suggestions on almost every aspect of the study.

References

[1] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[2] H. Chen, A. Li, G. Challen, and K. Cunningham. Implementation of split deadlines in a large cs1 course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2024, page 193–199, New York, NY, USA, 2024. Association for Computing Machinery.

[3] E. B. Cloude, R. S. Baker, and E. Fouh. Online help-seeking occurring in multiple computer-mediated conversations affects grades in an introductory programming course. In *LAK23: 13th International Learning Analytics and Knowledge Conference*, LAK2023, page 378–387, New York, NY, USA, 2023. Association for Computing Machinery.

[4] S. Dawson. Online forum discussion interactions as an indicator of student community. *Australasian Journal of Educational Technology*, 22(4), 2006.

[5] B. Erickson, S. Heckman, and C. F. Lynch. Characterizing student development progress: Validating student adherence to project milestones. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 15–21, 2022.

[6] Z. Gao, B. Erickson, Y. Xu, C. Lynch, S. Heckman, and T. Barnes. Admitting you have a problem is the first step: Modeling when and why students seek help in programming assignments. *Proceedings of the 15th International Confer-ence on Educational Data Mining*, 2022.

[7] Z. Gao, B. Erickson, Y. Xu, C. Lynch, S. Heckman, and T. Barnes. You asked, now what? modeling students' help-seeking and coding actions from request to resolution. *Journal of Educational Data Mining*, 14(3):109–131, Dec. 2022.

[8] Z. Gao, S. Heckman, and C. Lynch. Who uses office hours? a comparison of in-person and virtual office hours utilization. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2022, page 300–306, New York, NY, USA, 2022. Association for Computing Machinery.

[9] Z. Gao, C. Lynch, S. Heckman, and T. Barnes. Automatically classifying student help requests: A multi-year analysis. In I.-H. Hsiao, S. Sahebi, F. Bouchet, and J.-J. Vie, editors, *Proceedings of the 14th International Conference on Educational Data Mining*, pages 81–92. International Educational Data Mining Society, 2021.

[10] N. Gitinabard, Z. Gao, S. Heckman, T. Barnes, and C. F. Lynch. Analysis of student pair teamwork using github activities. *Journal of Educational Data Mining*, 15(1):32–62, Mar. 2023.

[11] W. Griffin, S. D. Cohen, R. Berndtson, K. M. Burson, K. M. Camper, Y. Chen, and M. A. Smith. Starting the conversation: An exploratory study of factors that influence student office hour use. *College Teaching*, 62(3):94–99, 2014.

[12] M. Hoq, J. Vandenberg, B. Mott, J. Lester, N. Norouzi, and B. Akram. Towards attention-based automatic misconception identification in introductory programming courses. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, page 1680–1681, New York, NY, USA, 2024. Association for Computing Machinery.

[13] J. R. Hott, M. Floryan, and N. Basit. Towards more efficient office hours for large courses: Using cosine similarity to efficiently construct student help groups. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2024, page 1684–1685, New York, NY, USA, 2024. Association for Computing Machinery.

[14] S. A. Karabenick. Classroom and technology-supported help seeking: The need for converging research paradigms. *Learning and instruction*, 21(2):290–296, 2011.

[15] S.-H. Ko and K. Stephens-Martinez. What drives students to office hours: Individual differences and similarities. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 959–965, New York, NY, USA, 2023. Association for Computing Machinery.

[16] M. Liffiton, B. E. Sheese, J. Savelka, and P. Denny. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, pages 1–11, 2023.

[17] T. M. MacWilliam and D. J. Malan. Scaling office hours: managing live q&a in large courses. 2012.

[18] S. Marwan, A. Dombe, and T. W. Price. Unproductive help-seeking in programming: What it is and how to address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 54–60, New York, NY, USA, 2020. Association for Computing Machinery.

[19] K. Nigam, A. McCallum, and T. M. Mitchell. Semi-supervised text classification using em. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, Adaptive Computation and Machine Learning, pages 31–51. MIT Press, 2006.

[20] A. J. Smith, K. E. Boyer, J. Forbes, S. Heckman, and K. Mayer-Patel. My digital hand: A tool for scaling up one-to-one peer teaching in support of computer science learning. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 549–554, New York, NY, USA, 2017. Association for Computing Machinery.

[21] M. Zahn, I. Gransbury, S. Heckman, and L. Battestilli. Assessment of self-identified learning struggles in cs2 programming assignments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, page 264–270, New York, NY, USA, 2023. Association for Computing Machinery.

[22] M. Zahn and S. Heckman. Observations on student help-seeking behaviors in introductory computer science courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2023, page 1380, New York, NY, USA, 2023. Association for Computing Machinery.

[23] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794, 2019.