WolfTutor

A system to enable peer tutoring built on Slack

Monica Metro
NC State University
mgmetro@ncsu.edu

Zachery DeLong
NC State University
zpdelong@ncsu.edu

Zhangqi Zha
NC State University
zzha@ncsu.edu

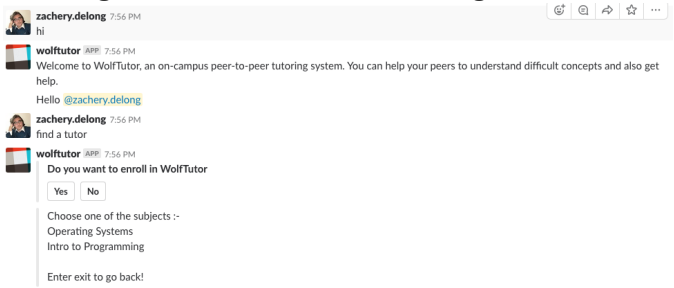
ABSTRACT

In this paper, Group-P presents an improvement on WolfTutor, a Slack-based platform for connecting students and facilitating peer tutoring. Group-P's work adds a recommendation module on top of WolfTutor's student-tutor matching system that attempts to find a tutor that fits into the using student's desires. A two-pronged approach to validation is also enacted where the system's algorithm is put through basic validation and the ease-of-use of the system is evaluated using in-person time trials with specific goals in mind. Finally, the group discusses the results of the evaluation and discusses future validation and improvements that will need to be done to make the application robust enough to use in a live educational environment.

1. INTRODUCTION

1.1 WolfTutor

Figure 1: The WolfTutor starting screen



WolfTutor is a system that seeks to enable students to tutor other students in a course-setting. It is a slack-based chat app that attempts to connect potential tutors in given subjects with students who need help with those subjects. Put another way, WolfTutor is an applicatio nthat focuses entirely on enabling peer tutoring. At first blush, it seems like the app is an app to actually facilitate peer tutoring, which is not entirely accurate. WolfTutor has functions to register tutors and students and to schedule tutoring sessions. It does not have functionality to actually perform the tutoring itself, but is a logistic tool to enable the coordination required to schedule tutoring sessions.

WolfTutor is also gamified. It rewards tutors who are highly rated with a points system which can be implemented in a number of different contexts to help incentivize students to tutor other students.

There are a number of things that WolfTutor does extremely well, chief among them its novel interface. Being a chatbot makes the application easy to port to new languages and new devices, since its entire UI consists of simple english sentences and some rudimentary web inputs like text boxes and drop-downs. That being said, there are a few potential areas for improvement, chief among them scheduling and tutor-student matching.

1.2 Literature Review

1.2.1 Peer Tutoring

While the merits of peer tutoring are not the major focus of this paper, it is prudent to give a brief overview of the topic. Peer tutoring is a type of tutoring where a student seeks out instruction from another student who has already studied the subject that the student is interested in learning more on. There are a number of terms for the tutor in this situation such as "mentor" and "proctor" but for the purpose of this paper we will simply use tutor to refer to the student-tutor and student to refer to the student seeking help. [4]

There are a number of different well-documented benefits to peer tutoring, and they are not limited simply to the students being tutored. While the intention is to improve the students first, the process of teaching another student has many well-studied benefits for the tutor as well such as enhancing understanding. [4]

While it is absolutely true that tutors benefit significantly from teaching their stduents, the obvious goal is to transfer knowledge from the tutor to the student. It is well documented that students tend to feel more engaged during one-on-one peer tutoring and that the peer tutoring can give more feedback than lecture-style learning which can in turn help reduce student anxiety and improve learning outcomes. [5]

1.2.2 Comparisons to Existing Applications

The application of WolfTutor is not actually specific to tutoring entirely. It can reasonably be compared to any

PLDI '13 June 16–19, 2013, Seattle, WA, USA
ACM ISBN 123-4567-24-567/08/06…$15.00
DOI: 10.475/123_4

scheduling system. For example, the appointment scheduling tool within NC State University's epack system functions similarly. First you pick from an available list of appointment types. Then, you can opt to filter your search further by location, names of possible appointees, time of day, etc. The tool will show available appointment time slots a user can book just like WolfTutor. This type of scheduling mechanism is commonly used by other services as well e.g. medical facilities with multiple practitioners, personal trainers, life coaches, etc.

There are no mainstream applications relevant to automative tutor matching that give the users tutor recommendations. Most services only offer a place to find tutors manually, such as the 'Lessons' section on Craiglist. [2] There are services with more detail and structure than Cragislist like Verbling, an online application that contains profiles of Spanish tutors. These profiles contain more detailed and personal information about the user, including a photo. [3] Some services take other approaches. The popular tutoring service, Chegg, does not facilitate scheduling or tutoring sessions, but instead lets multiple tutors cater to a student's question. [1]

1.3 Original System

Wolftutor has two types of users: students and tutors. Users are automatically students at signup, but can choose to become a tutor after following a simple prompt and giving some information about the topics they are interested in, the times they are available, and some academic information. After enrolling in the system, the user's name, email, and phone number will be taken from slack and stored if available.

1.3.1 Original Student Use Cases

1. Find a tutor by from a complete list of all available tutors after selecting a subject from an existing list
2. See the reviews for at tutor
3. Book a tutor by choosing one of WolfTutor's defined 30 minute slots created from the availability given by the tutor
4. Review a tutor from their last session (students have until the end of the day to review their tutor)
5. View active reservations
6. View reward point balance (points are used to schedule sessions; 100 points are given at signup)
7. Buy more points

1.3.2 Original Tutor Use Cases

1. Become a tutor. WolfTutor only allows one major and one degree per tutor, but a tutor can represent multiple subjects. Tutor self-determines points pay rate and availability. Both apply to all subjects.
2. View tutoring subjects given to WolfTutor
3. View tutoring availability given to WolfTutor
4. Redeem points commerically

1.4 Possible Enhancements

1.4.1 Enhancements

The original creators of WolfTutor proposed three additional features for future development. The first was to integrate an online platform such as Skype or Google Hangouts to conduct tutoring sessions online. This has obvious benefits for students because tutoring sessions could be conducted anywhere without needing physical proximity. For some subjects, especially Computer Science or programming subjects, this could work quite well, but for other subjects such as math it would be difficult to implement well without specialized hardware to allow handwriting to be digitized in real time or good cameras to share physical paper being written on.

The second enhancement idea was to allow both the tutor and the student to sync their session reservation with their calendar, e.g. Google Calendar or Outlook. This has the obvious benefit of keeping the tutoring session front-of-mind, assuming that the students actually use such platforms. It could also be possible to create a print-friendly view for paper calendars. In the end, the goal of this enhancement is to help students and tutors keep their tutoring sessions.

Lastly, to update the scheduling algorithm such that a user could edit or delete reservations in addition to being able to create and view them. Our team provided two other features: increasing matching options between tutors and students and allowing students to browse their reservation history.

After careful consideration, three main pain points were decided upon that consisted of the proposed enhancements. Integration of an online platform was discarded because it was not thought to facilitate the quality of the match between tutor and student.

Scheduling and Calendar Sync.

WolfTutor currently only allows users seeking a tutoring session to create and view reservations. In real life, people change their mind or events come up such that a schedule change is in order. The ability to cancel or reschedule reservations would make WolfTutor more applicable to real world scheduling scenarios. In addition, facilitating intergration with commonly used calendars such as Google and iOS Calendar extends this principle of making scheduling easier.

Increasing Matching Options.

In regards to matching with tutors, WolfTutor currently displays a list of tutor's and their ratings based on the student's desired subject. Then, the student may attempt to book a reservation with a tutor they choose. By expanding matching options, a student should be able easily find a higher quality match with a tutor. For example, a student could filter tutors by location by selecting only locations they want to meet up at in a checkbox. Similarily, if a student has a few tutors they prefer, they could select those names from a checkbox such that WolfTutor only displays those tutors. This type of filtering system is often used by medical facilities with multiple practioners and multiple practing sites. It is also used by NC State University's scheduling tool on epack.

History and Recommendations.

WolfTutor does currently allow for student's to review and rate the tutors that they met with previously. However, there is no way to view a student's reservation history past the most recent tutor. WolfTutor also does not provide a way for a student to easily re-book a new reservation with a tutor they chose previously if they liked the tutor and would like to schedule a reservation with them again. Adding these enhancements would increase ease of use by helping a student distinguish between a competent and non-competent tutor they've had in the past.

1.4.2 Initial Survey

A survey was conducted to determine which enhancement the intended user base (students) prefered the most. The survey was separated into three parts.

Background.

A background section measured how prevalent scheduling was within the daily life of the participants. Most participants admitted having to schedule a meeting within the last year.

Priorities.

The next section revealed which enhancement the participants thought was a priority by asking them to rate the level of important the enhancement was on a scale from 1 (least important) to 5 (most important). To avoid bias, instead of explaining the enhancements out, the survey questions were created around the base point of the enhancement:

- "When scheduling a tutoring meeting, picking the location is very important to me."
- "When scheduling a tutoring meeting, the competency of the tutor is very important to me."
- "When scheduling a tutoring meeting, being able to agree on a time quickly and easily is very important to me."

The question regarding competency scored the highest level of important among the participants collectively. Scheduling and increased matching (by location) followed respectively.

Trade Offs.

The objective of the third section was to validiate the results in the second section by asking the participants which enhancements they would be willing to compromise on in order to get the enhancement they thought was more important. The questions included:

- "When scheduling a tutoring meeting, I am willing to make trade-offs on the competency of the tutor and time if I can specify the location of the meeting."
- "When scheduling a tutoring meeting, I am willing to make trade-offs on location and time if I can specify the competency of my tutor."
- "When scheduling a tutoring meeting, I am willing to make trade-offs in location and competency if I can specify the time of the meeting."

Again, competency scored the highest as the most important enhancement collectively. Scheduling followed in second place and location matching in third. See Figure 2, Figure 3 and Figure 4.

Figure 2: Initial Survey Trade offs: Competency

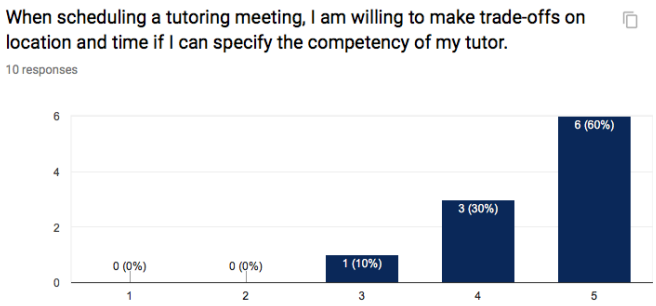


Figure 3: Initial Survey Trade offs: Location

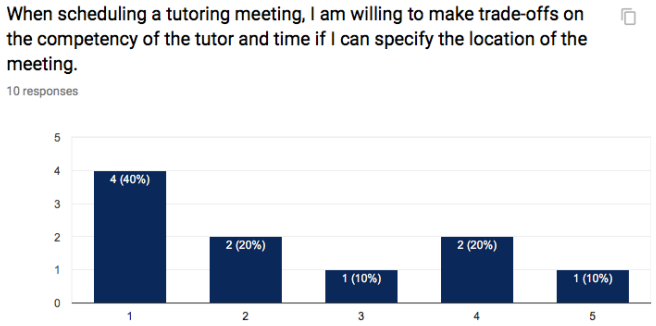
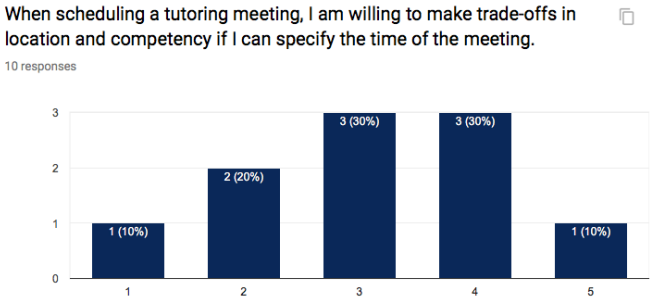


Figure 4: Initial Survey Trade offs: Time



Other.

An option was given to participants to suggest a new enhancement. The only received response was to integrate the application with Skype.

1.4.3 Chosen Enhancement

This pre-surveying is something the group did not originally intend to perform, but decided to do because the team was deeply split on what enhancement(s) would add the most value to students. After performing the survey, however, the team found that the sample of student surveyed

heavily leaned toward enhancements that were not being seriously considered at the time.

Since the vast majority of surveyed users from our target audience elected easier matching with competent tutors, our chosen enhancement focuses on tutor matching with regards to the qualities of the tutor that make the tutor a good tutor. This enhancement is explained in more detail within section 1.5.

1.5 Tutor Matching

WolfTutor's existing mechanisms for matching students to tutors are fairly rudimentary. When tutors register for the application, they are asked to give a set of subjects they are comfortable helping on (the list of which is determined and maintained by system administrators) and a list of days and times they are available to instruct.

While immediately useful, this is not nearly as far as the system could go in terms of matching students with tutors. The team has built out a mechanism for enabling the matching of tutors on multiple criteria and has created an easy pathway to add or change the criteria easily.

One major concern for the team during development was the choice of a recommendation mechanism. While the team knew immediately that adding some kind of recommendation algorithm was going to be necessary, the actual mechanism is something that was debated significantly. In the end, the team opted for Occam's Razor: the simplest algorithm possible to create the most value possible, which in the future could easily be replaced or enhanced to compare performance.

The tool the team chose was a simple weighted average. The process is fairly simple: several criteria were chosen to each generate a respective "score". That score is then assigned a weight, and each tutor's score is averaged together with these weights, which can then be normalized and used to rank tutors. In this way, the problem of recommendation becomes a sorting problem, which can easily be solved using a number of very fast algorithms.

This approach also has the advantage of leaving each tutor with an individual score which can be used as input to a number of other possible algorithms, which will be discussed in section 4.

2. DESIGN

2.1 Enhancement

As mentioned in section 1.5 the goal of this project is to improve the matching done by WolfTutor. In its original form, students input only what subject they were interested in a tutor for and the system returned back all students in that given subject. While effective, this brute-force strategy can wind up showing students an extremely large number of potential tutors in their chat app, which can be challenging to navigate.

The original system also displayed fairly little information about the tutor when searching. The list of tutors returned by the "find a tutor" functionality contained basic information such as name and major, but did not include information about the performance of the tutor academically or their reviews. To see reviews, the searching student would have to hit a "review" button which would then display all the reviews that tutor had at the bottom of the screen. While this technically made the necessary information available, this UI was challenging to use (requiring significant scrolling back and forth through the history in a chat app, something usually not encouraged in chat bots) and after getting feedback from the first round of reviews, the team decided to make some changes to this process.

First and foremost, however, the goal of this project is to improve the process of chosing a tutor in WolfTutor by providing a filtered list of tutors to students when they search for tutors in the system. To do this, the team identified four major metrics of a tutor to use as a basis of comparison. The primary nonfunctional requirement of this enhancement is to keep the tutor acquisition process streamlined and fast. That means any enhancement will have to be evaluated for speed of use, something we will discuss more in the evaluation section

2.1.1 Overall Review Score

The overall review score is the most obvious metric for evlauating a tutor currently. This score is simply a mean of the reviews (on a scale from 1 to 5) of a given tutor. A simple average, though, has a few problems.

- Tutors who have used the sytem for a longer period of time are less sensetive to fluctuations in their rating and a dramatic change in quality of tutoring may not affect them appropraitely.
- Tutors who had a temporary period of poor reviews or a single bad experience early in their career may struggle to find students to redeem their record after a small number of poor reviews.

Given the potential for abuse, the team decided to break the reviews into a rolling thirty day window. By keeping reviews to the past thirty days, the team hopes to help prevent some of the potential abuses of the mean review score by allowing outlier reviews to fall off on a regular schedule. This also serves as a small optomization to the review calculation process to help keep the calculation of the mean scores fast enough to scale in a production system, one way to help start to satisfy the performance nonfuncitonal requirement.

It is also worth noting that this window is configurable, so if a problem were found during evaluation, this is one dimension that the system could be tweaked to help prevent abuse.

2.1.2 Individual Review Score

The individual review score is one dimension that offers personalization to the reviews. While it is obviously a good idea to order tutors on the basis of their overall review rating, it is also possible that particular tutors work well with particular students. To that end, any system making personalized recomendations should take into account the searching student's preferences and past experience. This is what the individual review score attribute attempts to do. This score is the mean review that the current student has given to each tutor.

2.1.3 Tutor Score

The tutor score is similar to the individual review score. Like the individual review score, it is a score specific to the logged in student, but instead of being the mean review score the current student has given a given tutor, it is the mean review score a given tutor has given to the current user. This attempts to solve two problems.

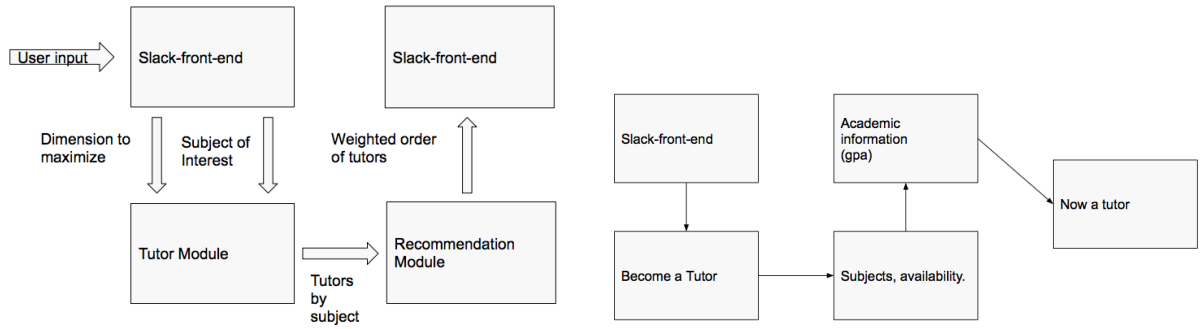


Figure 5: The flow of the "request a tutor" use case (left) and the new tutor registration (right)

- It helps new tutors rise in the rankings. It helps tutors with short review history but strong experience being tutored show up in recommendations.
- It helps further personalize recommendations.

It is the team's hope that this will help encourage tutors to seek out tutoring from the peers they have been tutoring when they use the system to find help in their subjects. This also may help encourage students who have been using the system to become tutors and have a leg up if they worked well with a tutor in the past. In the current iteration of the software, this attribute is weighted fairly high by default, but this is one of the dimensions for customization and could be tweaked easily.

2.1.4 GPA

The student's GPA is the last dimension the team found time to add, and it needs little introduction. At some universties such as Marshall University, students are not authorized as peer tutors unless they have attained a GPA better than some threshold. While WolfTutor does not currently employ such a GPA filter, it is possible that such a minimum viable threshold could reveal itself in whatever culture the application was deployed to. It is also possible for universities to set their own minimum value in the future as the application is deployed to meet with existing university policies.

2.2 Re-weighting

One problem that revealed itself with a simple weighted average almost immediatly is that a student who performs poorly shows up in the list ahead of a new tutor. After doing some initial testing, the team desired to tackle this problem by rescaling the three review scores (overall, individual, and tutor) on a scale between -2 and 2. To do this, a function was implemented that maps scores between 1 and 5 are subtracted by 2, while scores of 0 are left unchanged. This means that a tutor with a low score will be penalized for their low score, but new users are not penalized similarly.

2.3 Bugs

The existing code was of a reasonable quality at the time that it was handed off to group-p. That said, there were a few issues that needed to be addressed before work could proceed.

The first and most glaring is in the documentation. The documentation provided to group-p was actually quite good. The team was able to get an application up and running with the basic specification within a few hours, but one step was left out that prevented any registration as a tutor or searching for tutors: a database entry had to be created for at least one subject manually. Once the team reached out to the original developers of WolfTutor, the problem was rectified within 24 hours and the appropriate documentation was updated.

The next bug that had to be fixed was in the review process. For whatever reason, the exisitng review functionality was nonfunctional when tested on group-p's development environments, and some time had to be spent getting that functionality working for testing.

Lastly, there was a very minor issue that would cause a timeout message to appear every time the bot was restarted and a new student would start interacting with it. This bug did not break the software, but the team did spend a little time to root it out just the same.

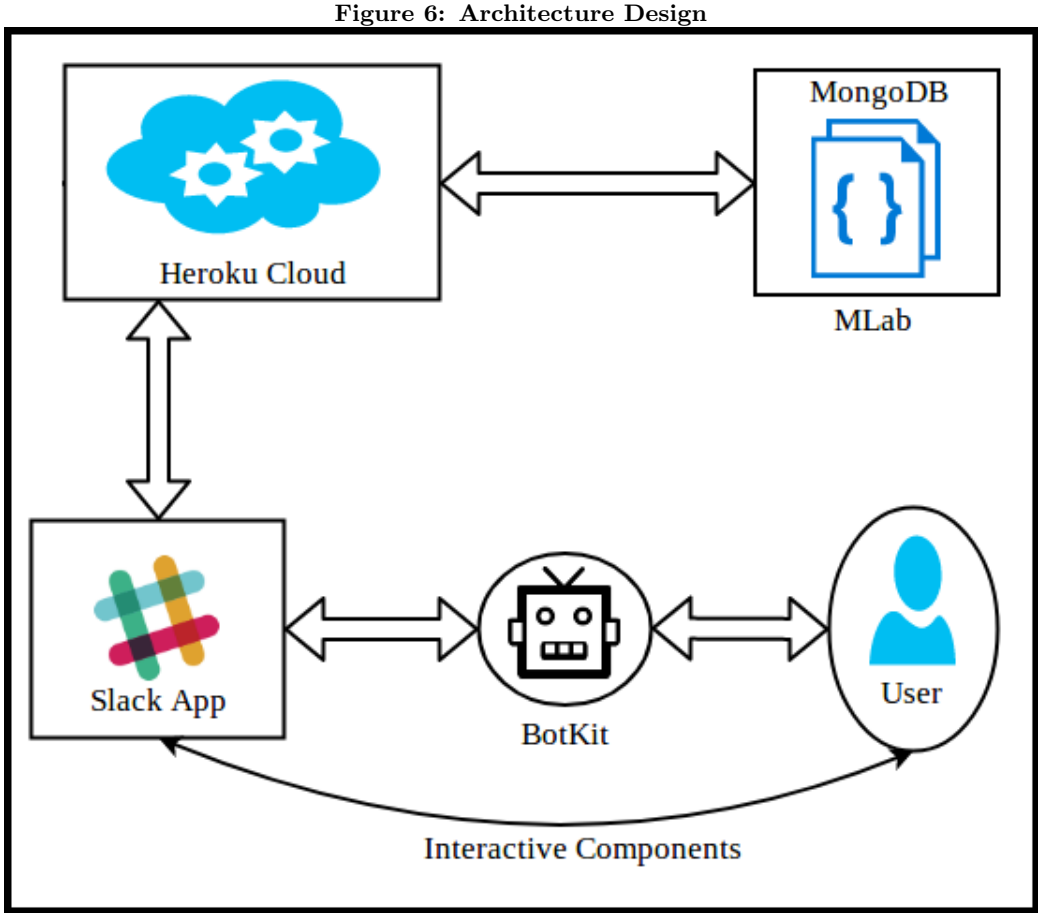
2.4 New and Updated Use Cases

After integrating the enhancements to the application, the following changes were made to existing use cases as defined in sections 1.3.1 and 1.3.2.

1. Viewing past reservation history instead of only actively scheduled reservations
2. Tutors are required to list their GPA for their given degree. Users are able to view that GPA when searching for a tutor
3. Viewing the average review score of tutors when searching for a tutor
4. Allowing the user to opt to prioritize GPA or review scores for the matched tutors such that the user will see tutors filling those criteria first
5. Tutors are displayed in reverse order such that a user will see the most recommended first in the Slack Application. Originally, a user would have to scroll all the way up to where the command was given to find a tutor to find the first posted tutor

2.5 Architecture

Figure 6: Architecture Design



2.5.1 *Overview*

Figure 2.5.1 outlines the architecture of the application. There are three main divisions: The user interface, the application server, and the database. Slack was used as the user interface. A custom bot was installed to that Slack page to handle interactive elements with the users. The applicaton was built with NodeJS. MongoDB was the database chosen to hold user data. For development purposes, a local server of NodeJS was run along with a local mongoDB database. Ngrok was used to tunnel webhooks between our local servers, which was necessary to get past local NATs and firewalls securely. The specific infrastructures are discussed as followed.

2.5.2 *Slack*

Slack is primarily a business tool intended to help facilitate communication and coordination between individuals and groups at organizations. In practice, however, Slack has become a familiar name with many unintended audiences such as online communities and even university courses such as this one because of it's availability and ease of use. Attracting users to a platform they're already familiary with is generally easier and less costly than attracting attention to a brand new website.

Another reason for using the Slack platform is that it already supports the languages and keyboard layouts for English, Japanese, German, Spanish, and French. Their website also claims to be including more regions in the future. Because of this, using a platform like Slack is beneficial to smaller developers as they can spend less time building a unique website and pushing it out internationally and more time on the functionality of the product.

Slack Bot.

As mentioned previously, the target for this chat bot is to run in the Slack platform. The Slack bot handles the interactions between the user and the NodeJS applications. All responses given by the user would be posted to the application by the bot. Similarily, all prompts sent to the user from application would be posted to Slack by the bot.

While it is true that this bot has been deployed to Slack specifically, it was developed using a popular API called Botkit, which actually has APIs for multiple chat services such as Facebook Messenger or Discord as well as Slack. Because of this, it is possible to port WolfTutor to these other platforms without needing to entirely start over.

2.5.3 *NodeJS*

NodeJS is the primary framework being used in this project. Node is a runtime for Javascript which has proven to be strong competition for more traditional LAMP stacks in the web. In this case, it provides a number of different libraries which make development of a chatbot easier. Running Javascript also has the benefit of making the chat bot easier to work on for a large number of people given Javascripts ubiquity in recent years.

Main.js.

Any interaction with the application via text is handled in the "main.js" module. In this module are event listeners for text entry events in the Slack chatbot. Any time a student is typing a response or initiating contact with the bot for the first time, the events for handling their inputs are in this file in a series of listeners and callbacks. Interestingly, the Botkit API used for this project does not natively support a more modern JavasScript promise-based design.

Dialogs and Prompts.

Dialogs are the other main way that students interact with the system. In Slack, a dialogue is a pop-up window that appears over the chat window and contains web-controls such as text boxes, radio buttons, or drop-down select boxes. These dialogs can be sent to the chat bot through the responses to user's input in the "main.js" module, and can provide a richer interface for providing structured data such as selecting subjects from a list or time slots in a calendar. The dialogs themselves also have a "message response" identifier which will be sent back to the slack bot when the user responds which is the other primary way users interact with the system. These response identifiers provide an easy way to re-order or re-use messages without having to make large code changes. Interestingly, slack only allows 5 controls to be in a single dialogue at one time.

Modules.

The last major place for code is the modules folder. The database access is handled through the models but helpers are built around complex tasks and stored in appropriate JS classes in the modules folder. for example, the actual database manipulation for tutors happens in the tutor model, but a tutor module also exists that utilizes that module and adds business logic to it.

2.5.4 MongoDB

MongoDB has serveral advantages. It is well documented for use with NodeJS, it is easy to set up a local database, it is also easy to set up online with MLAB platform and integrate into Heroku, and it is also easy to access through authorization credentials. Creating and uploading a mock database with a python script only took seconds with MLAB's authorization abilities to allow us to access the database outside of the MLAB site portal.

2.5.5 Heroku

Heroku is a popular cloud platform service that enables developers to run their applications online. Because of its popularity, there is an ample amount of documentation for using Heroku among other services such as NodeJS and mongoDB. Heroku also offers a free personal server that was used to speed up in person evaluations of WolfTutor. To start the server, the single free dyno of the application's server would execute the Procfile via the 'npm start' command.

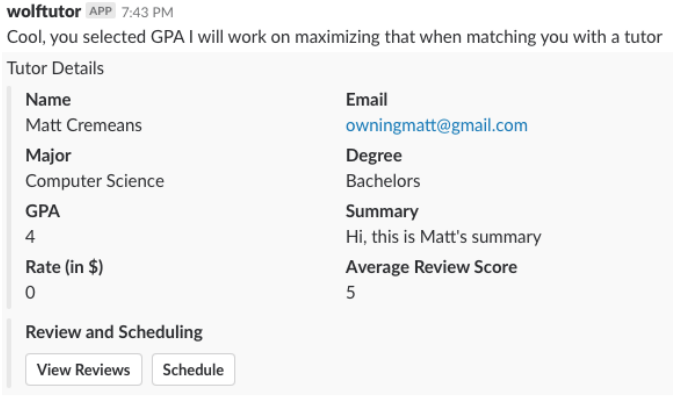
3. EVALUATION

3.1 Post Survey

3.1.1 In Person Evaluations

3.1.2 Responding to Change

Figure 7: The tutor display prompt for a volunteer.



The team performed evaluations in person and over two days. At the end of the first day, there were a few issues that had been consistently pointed out that were fairly straightforward to fix. To help focus on evaluating the speed of use of the system, the team opted to fix a few of these issues immediately before the second round of tutoring. In addition to these changes, section 3.2 discusses slight alterations made to the databases after evaluation day 1.

The first thing that was changed was a menu from the original system. In the original system, the tutor display had a "Rate" section that was intended to display that tutor's billing rate. During our trials, testers consistently mistook this for a rating, so the team made the decision to add the quantifier added in figure 7 switch things out.

The team also decided to add one other menu item. During the initial development, the team intended to add average tutor review score to the same tutor display discussed above, but overlooked that simple enhancement at the time. This made it more difficult for students to actually run the tests, though it was still possible through the use of the original "review" view. This field is also visible in figure 7.

The final major change made after the first round of tutoring was a bugfix. During the first round of evaluations, the team realized that the tutors being displayed were displaying out of order from what the team expected. After investigating, it was discovered that the suggestion algorithm was not malfunctioning as originally thought, but that the Slack API used to send the tutor suggestions to the users was delivering messages asynchronously, rather than synchronously as had been thought during development and initial testing. This was rectified by changing to a different interface that, while slower because it performed a handshake with every message, delivered the messages in the correct order.

3.2 Creating Mock Tutors

3.2.1 Random Data Creation

To evaluate WolfTutor without bias, random mock tutors were created via functions from python's Random library, except for tutoring subjects as all tutors could tutor in the four possible subjects, names which were generated as User N where N was the number of the user in creation order, and emails similarily being userN@ncsu.edu. Two separate databases were created for both the new WolfTutor appli-

Figure 8: GPA Distribution with 1000 Tutors

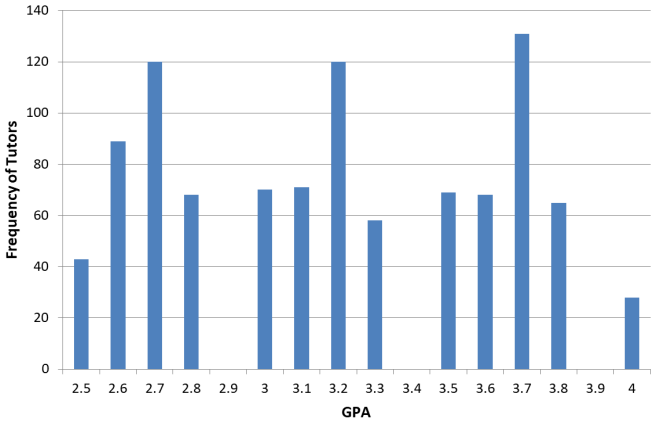
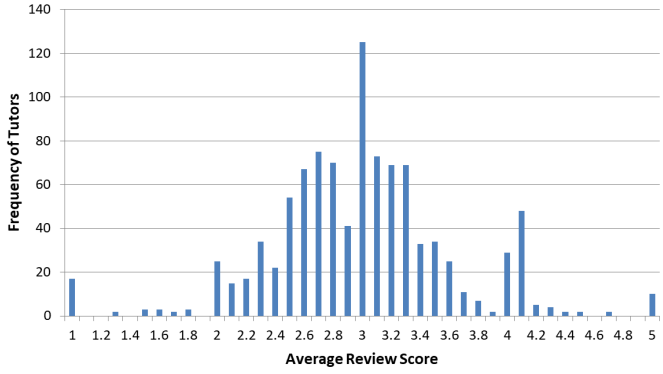


Figure 9: Average Review Score Distribution with 1000 Tutors



cation and the old, to prevent reviewers from automatically looking for a tutor they liked when moving from one application to another.

Originally, majors and degree level were chosen randomly, but during in person evaluations, many reviwers voiced that they did not see what they were looking for. For example, one reviwer specifically wanted a Computer Science major with an extremely high GPA but the highest GPAs belonged to other majors. Another reviwer wanted only tutors with master degrees and felt that they did not have many options. For the second day of evaluations, instead of creating more options, new databases were created such that all majors were Computer Science and all degrees were Masters. Adding more options would have created more load than the application was currently able to handle. The second day reviwers were noticeably more content and less confused.

3.2.2 Random Data Distributions

During day 1 in person evaluations, it was noted that there seemed to be one or two 'God' tutors that were popular. With further inspection, it was determined that the database was a bit too random and needed to be altered slightly to avoid a situation where a small number of tutors were obviously better than the others in order to better test the time needed to use the application more realistically. For example, not allowing the tutor with the highest GPA to also have the best average review score.

This issue was investigated by creating histograms for the distributions of both GPA and average review score over 1000 tutors. Figure 8 is the distribution of GPAs created for one test. Additional tests created nearly identical distributions. Figure 9 is the distribution of average review scores for one test. Additional tests created nearly identical distributions in this case as well.

The GPA distribution is discrete uniform as it should be. However, tutor GPA is not best represented as uniform because if someone has a very low GPA, they likely don't know enough about the subject to tutor it. Students also typically don;t want a tutor with a bad GPA. Because of this, some GPAs between 1.0 and 2.0 were raised to between 3.0 and 4.0 for day 2 evaluations.

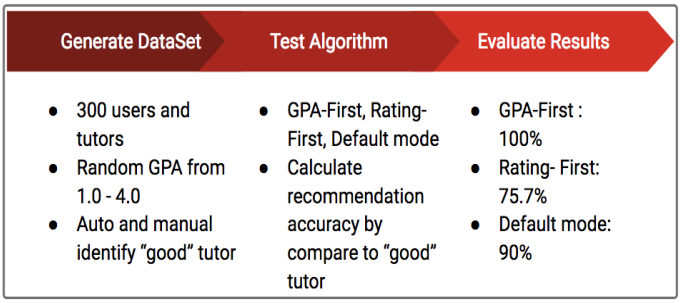
The average review score distribution is not uniform because it is calculated by taking the average of the individual scores given generated as discrete uniform distribution. Taking the average created an almost normal distribution. This seemed to work well for day 1 evaluations, however, some alterations were made to the day 2 database to create more competition between the tutors and to avoid 'God' tutors previously mentioned.

3.3 Algorithmic Evaluation

As we implement the enhancement of wolfTutor during each sprint, a variety of tests were conducted to verify the functionality and performance. For example, 1) unit tests were generated after the implementation of some core part, like prioritize method, so that to make sure the functionality of the new feature. 2) We also conducted a full user case manually tests after each sprint, to verify the functionality of the whole application. This means we fire up the server, mock the process by enrolling in the system, becoming a tutor, switching to another user account, finding a tutor, reviewing a previous tutor session, and also loading the history of previous tutor sessions.

In order to verify tutor matching algorithm quantitively, we define tutor suggestion accuracy as the percentage of how many the suggested tutors are in the good tutor set. The good tutor set was generated manually or automatically from a serial of matched tutors for a certain user, and this list of tutors will be considered as labeled good tutors. Then we ran our matching algorithm, logged the output and result into json files. Finally, we compared the output with the labeled good tutors and computed the accuracy.

Figure 10: Quantitive Evaluation Process



Generate Dataset.

We need a database which contains a fair mont of user and tutor to work with. Manually adding users and tutors through the slack app interface will be too time-consuming. Instead, the python script described in section 3.2 was used to automatically generate a fixed number of users and tutors, along with their randomly assigned attributes. For the quantitative testing, the most important attributes are the ones related to tutor experience and academic performance - review scores and tutor GPA. These attributes are directly used in the matching and suggestion algorithms.

In this quantitative verification, we used that script to generate 300 users and 300 tutors saved into two separated json file. Each tutor was assigned a random number of reviews with random rating ranged from 1- 5. Each tutor also was assigned a random academic performance - GPA with a number ranged from 1-5.

Test Design and Execution.

We designed three major test cases to evaluate the suggestion algorithms. 1) GPA-First: while a user was searching the tutor, they may demand some tutors with high academic performance, here we use GPA as the indicator. In this case, the suggestion algorithms should output a list of tutors which was GPA dominate the order. 2) Rating-First: the same scenario except that user may demand tutors with high tutoring performance, the history rating they got from previous sessions. 3) Default mode: user does not have any preference, but will get a list of suggested tutors with a weighted average score of all kinds of metrics.

We implemented these test cases in Node js, executed along with all other test cases, and saved the recommended lists of tutors of each test cases into JSON files for more quantitive analysis.

Accuracy Result.

In order to analyze the accuracy of each scenario, we need to define the good tutor sets first. 1) GPA-First: we sorted the tutors by their GPA score, then picked the top 20 as the good tutor set. 2) Rating-First: we sorted the tutors by their average rating, then picked the top 20. 3) Defaut mode: we considered all the tutors with GPA higher than 3.0 and rating higher than 3.0 as the good tutor set.

After we identified the method to extract the good tutor sets, we coded in python script. This script can also do the extraction information from test result JSON file, then computed how many suggested tutors are in the good tutor set, this will be the accuracy for a certain scenario. We ran each scenario for each user and averaged the accuracy as the final results. Figure 10 show the results.

4. CONCLUSION

4.1 Future Work

4.1.1 Further Evaluation

The team has made a best effort to evaluate the algorithm presented in this paper both using a manually-labeled data set and using in-person interviews with potential users. That said, true evaluation of educational software must be done in a classroom setting.

A good way to evaluate the usefullness of this system would be to truly put it in the hands of students over a period of time. Over the course of several semesters, a different set of sections of one course should be evaluated. For this reason, it may make sense to use a popular core course to a given major, such as Intro to Computer Science. These sections could be grouped into 3 groups, one with no intervention, one where students are pushed to use the university-sponsored tutoring facilities that exist today, and one where students are given access to the same facilities but are also given access to WolfTutor. Interviews would need to be conducted after student tutoring sessions throughout the semester asking the students for feedback about the quality of their tutoring session specifically and features of the application more broadly. At the end of the semester(s), the grades of the different groups could be compared to historical data for the course and some generalizations could be made about the usefulness of the software.

It is important to note, however, that an improvement in grades may not be the right measure of success for this system. While better grades would certainly be ideal, simply higher rates of engagement between students and tutors would be one reasonable measure of success. Also, reducing time spent scheduling tutoring sessions between students in the traditional tutoring section and WolfTutor students would also be a reasonable win for students.

4.1.2 Future Enhancement

The other primary direction for future work is in the application itself. The recommendation module has been developed in isolation from the rest of the code base, so it could easily be plugged into another process with minimal effort. Whatever module simply needs to implement a method that takes in a list of tutors and most of the integration work should be quite minimal. The weighted average that has already been implemented in this paper can serve as a good baseline for future evaluation of other methods of classifying tutors. Other algorithms that might be a good fit for this problem are clustering algorithms or classification algorithms.

Clustering algorithms may be a good fit because they all amount to identifying groupings of similar entries in the population provided. This is not unlike what we want to do when searching for a tutor, so it might be possible to cluster students and suggest tutors that exist in the same clusters as students.

It is also possible that simple classification algorithms like logistic regression on the simple end and SVM on the more complex end could be used to classify tutors as good or bad. That said, performance is a major concern in a real-time application such as a chatbot, and such heavy algorithms as clustering and classification may be too complex to run reliably in a production environment.

Another direction for future work is identifying more dimensions on which to match tutors. Currently we are matching on three review-related scores and one academic performance score (gpa). This could easily be expanded by asking tutors and students more questions about what they are looking for. Does the student prefer test-taking or projects? Do they speak English natively or would they prefer to use some other language? The team intended to implement more dimensions to match on during the project, but time constraints forced limiting it to four. Fortunately, though, the application can easily be extended and adding more dimensions is a fairly easy future direction.

4.2 Final Thoughts

The final thought to conisder is whether or not group-p's contribution to WolfTutor is truly an improvement over the existing system. In most fields, the combination of the algorithmic evaluation provided in section 3 and the use validation provided in 3 would be sufficent to argue that the system is an improvement, but the education field is one that has higher stakes than most.

Educational data miners have spent considerable ink discussing the various pitfalls in identifying high and low performers algorithmically, and the problem of tutor recommendation is something that could potentially also be problematic. If a student is connected with a poor tutor or one that handles their position of power poorly, the potential impact could be catastrophic. This may help explain why many universities (such as Marshall University) require students to take the courses they tutor for and bar anyone with lower than a B in that subject from tutoring on top of requiring an in-person interview.

So does WolfTutor do enough to guarantee that a tutor is good or that they will work well with the students? The evaluation that has been done so far points that it will make a good-effort to make recommendations that will "do no harm" but it would probably still make school administrators uneasy. When combined with a university vetting process, though, WolfTutor, especially after the application of group-p's recommendations could serve universities and other levels of schooling well as a platform for connecting students to peer tutors.

5. CHIT

- RHO
- MEY
- NOC
- LJI
- ZSO
- CNM
- YMA
- YNH
- VJK
- PBZ

6. REFERENCES

[1] Chegg.

[2] Raleigh's craigslist lessons.

[3] Verbling.

[4] M. M. Kim. Peer tutoring at colleges and universities. *College and University*, 90(4):2–7, Summer 2015. Copyright - Copyright American Association of Collegiate Registrars and Admissions Officers Summer 2015; Document feature - ; Last updated - 2017-11-22; CODEN - COLUBY; SubjectsTermNotLitGenreText - United States–US.

[5] K. J. Topping. The effectiveness of peer tutoring in further and higher education: A typology and review of the literature. *Higher Education*, 32(3):321–345, 1996.