Predicting Wait Time for Computer Science Tutoring Lab

Patrick Hayes
University of California
San Diego
phhayes@eng.ucsd.edu

Abstract

The computer science tutoring lab has begun collecting a variety of tutoring data to help instructors, tutors, and students make more informed decisions. With over 50,000 tutor sessions recorded, they have been able to recommend the best times to add more tutors. They have been able to flag when a tutor is being underutilized, and they have been able to suggest when a tutor is spending too long with a single student. I hoped to add to this list by developing a model that can accurately predict how long a student will wait before a tutor helps them. This way students who have class in a half an hour do not wait needlessly if the expected wait time is 45 minutes.

Keywords — predicting wait time; tutoring analytics;

I. Dataset

Acquiring the Dataset –

I acquired the dataset by requesting it from the autograder development team. While the website is called "autograder", autograding is only one feature they provide. The website also provides a very popular online tutor queue. Previously students would add their name at the end of a list on the white board and a tutor would come by look at the next name on the list and help that student. Now students log on to their autograder account, select the class they need help with, and create a ticket which is placed on the queue for that class. The tutors then accept tickets and once they have finished helping the students they mark the ticket as resolved. This provides a record of how long the ticket took, who was getting helped and which tutor was helping them. All of this information is stored in a SQL database on the website's server. When I requested the data from the development team, they sent me a copy of the sql database.

Processing the Dataset –

There were many features I hoped to use for predicting wait time that were not readily available in the SQL database provided to me. For example, I wanted to know how many tutors were on duty at the time the ticket was created. However, the Ticket SQL table had no feature for tutors on duty. It didn't even have a field for the tutor who accepted the ticket. Instead there is another table called TicketEvents which recorded all actions done on a ticket. So when a ticket is created a new row is added to the Ticket table, but a new row is also added to the TicketEvent table to show the ticket was created. Then when a tutor accepts the ticket another row is added to the TicketEvent table to show the ticket was accepted and it records which tutor accepted the ticket. So how did I process the data to find my "tutors on duty" feature? I did a join between the Ticket table, the TicketEvent and the Queue table to get a history of all the actions taken on every ticket for a certain queue. I then sorted all the events in chronological order. I then traversed the history one event at a time. While I was doing this traversal I was keeping a list of all the tutors who had done some action in the last 30 minutes. If Sally Tutor had accepted a ticket 23 minutes ago I assumed she was still on duty when this new ticket was created. So for every step in the traversal I would check if this event was the creation of a new ticket, if it was then I would add a row to my dataset and also record the current size of my active tutor list.

I had to do similar traversals of the ticket history to acquire many other features which were not readily available in the original SQL database. After processing all of the data I had a json file with 54,143 tickets.

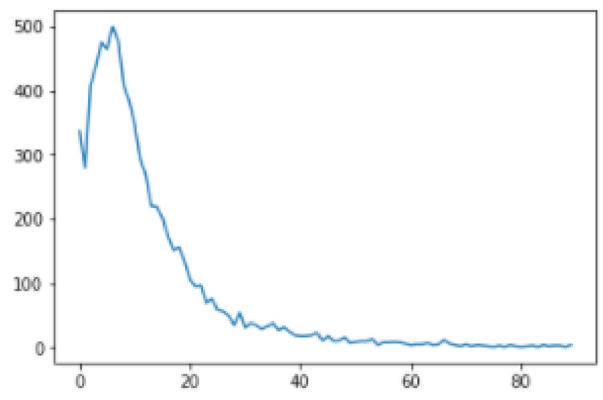
Distribution of the Data –

When exploring the data, I wanted to make sure I had an adequate number of data points for each feature I used. One of my fears was that by differentiating by course I would cause some features to have less than 100 tickets to train on. So far 10 different courses have used the autograder queue, but of those 10 some have used it more than others. For example, CSE12 has over 17,000 tickets, but CSE160 only has 103 tickets. Some features affect wait time differently for different classes, so I want to differentiate between classes. However, if some courses only have 103 tickets, those courses would be in danger of overfitting. To prevent some features from having less than a hundred tickets to train on I only have a feature unique to a course if that course has at least 5,000 tickets.

I also wanted to make sure each tutor had helped an adequate number of tickets to prevent tutor features from overfitting to just a couple tickets. 90% of tutors had helped at least 10 tickets, which makes sense. When a tutor tutors a course they usually tutor that course for an entire quarter which gives them plenty of time to help at least 10 tickets. I think 10 tickets is enough to get a sense of how fast a tutor takes to help a ticket compared the average, so I set 10 tickets as the cut off. If a tutor had helped less than 10 tickets I treated that tutor as an unknown.

As I was exploring the data, I also noticed that some tickets which should only take a couple of minutes to resolve based on their description were taking upwards of three hours. These tickets occurred at the end of the day, so I assumed the tutor simply forgot to resolve the ticket before they left for the day and the ticket was finally resolved at 4 am the next morning (when the queue closes). These outliers were skewing the average time to resolve a ticket and were heavily skewing the average time to resolve the last 10 tickets. To get rid of these outliers I wanted to use a trimmed the mean which only includes times that are within 2 standard deviations of the mean.

However, the mean is 12 minutes with a standard deviation of 16 minutes.



(Figure 1 – x-axis: time to resolve in minutes, y-axis: number of tickets)

If I used a trimmed mean that used anything within two standard deviations, I would use any ticket which took between -20 minutes and 44 minutes to resolve. The standard deviation metric assumes a normal distribution, but as seen in figure 1, the distribution for resolution time has lower bound. I decided to filter out any ticket which less then 5 minutes or more than an hour.

II. Predictive Task

Defining the Task –

Predict how long a student will have to wait before they are helped by a tutor.

Measuring Performance –

When measuring the performance of my models, I wanted to measure their ability to accurately predict the wait time for a normal ticket. Certain tickets are incredibly difficult to predict and there are potentially mislabeled tickets in the data set, so I wanted to use a performance measure that would favor models that are very accurate most of the time but very inaccurate every once in a while, over models that are moderately accurate all of the time. When comparing two models I compared the trimmed mean and standard deviation of their squared error and graphed their absolute error. I favored models with smaller means and standard deviations and used the graphs to make sure make sure outliers were not skewing the statistics.

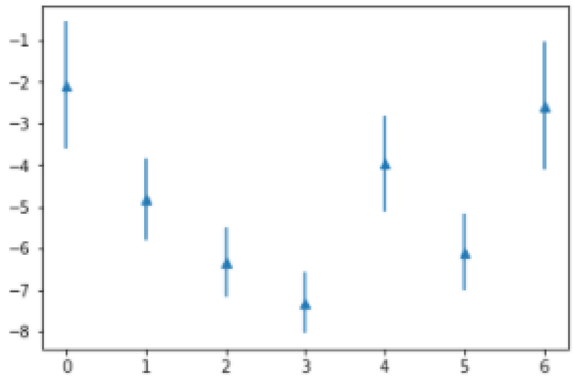
Choosing a Model –

I used linear regression to predict wait times. I compared the performance of four different models. The first model always predicted the average wait time. This model acted as a naïve baseline. The second model was a single layer perceptron with 31 parameters, the third model was a multi-layer perceptron with a total of 31 parameters, and the forth model was a multi-layer perceptron with 221. I wanted to compare a single layer perceptron with a multi-level perceptron with a similar number of parameters to see if there was a performance difference between the two. The final model had the most parameters and thus took the longest time to train and was the most susceptible to overfitting, but could also have more complex relationships between the features which increased its accuracy.

Choosing Features –

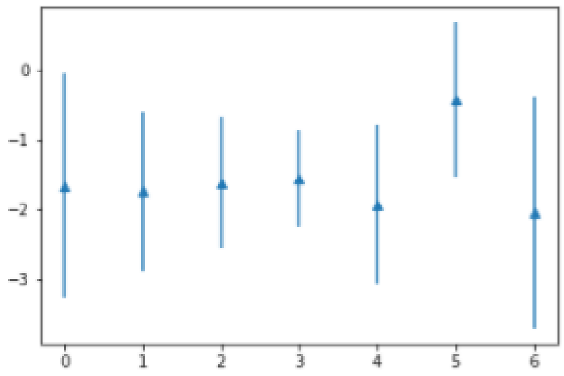
If you know how many tickets are in front of you in the queue, the number of tutors on duty, and the average time it takes to resolve a ticket, you have a very good idea of how long it will take you to get help. However, there are other features that can refine your prediction. For example, some tutors spend more time per student then others. If you know which tutors are on duty and how long the long they tend to spend per student, you can make a more informed decision. Likewise, the average time it takes to resolve a ticket can vary significantly between courses, so if you know which course this ticket is for you can improve your prediction. Even temporal features like time of day or day of week are correlated with variances in wait time.

I was confident that knowing a ticket's course would improve the wait time prediction, but I was skeptical about time of day and day of week. There was a correlation between day of week and wait time in the data set, but my suspicion was that the time of day and day of week were correlated with other features and it was those features that were actually causing the differences in wait time. To test my suspicion, I plotted the error between the actual wait time and my predicted wait time according to day of the week. When I only used number of tutors, position on the queue, and average time to resolve a ticket there was still a correlation between the error and the day of the week.



(Figure 2 – x-axis: day of week Monday-Sunday, y-axis: predicted wait time minus actual wait time)

Figure 2 shows the how the day of the week affects the prediction error for the CSE 100 queue. I ran 100 simulations on a random fifth of the data and plotted the mean and standard deviation of all the simulations. As you can see when the predictor only uses number of tutors, position, and the average resolution time there is still a correlation between wait time and day of week. However, if we include a term for tutor bias into our predictor and graph the error in terms of day of the week we get the following graph.



(Figure 3 – x-axis: day of week Monday-Sunday, y-axis: predicted wait time minus actual wait time)

Based off these graphs and other similar graphs for time of day I decided that day of week and time of day were weak features for predicting wait time and decided to remove them.

Final List of Features –

- Position on queue
- Number of tutors on duty
- Course
- Tutor bias (explained in paragraph below)
- Average time to resolve the last 10 tickets (explained below)

Tutor Bias Term –
To calculate each tutor's tutor bias term, I first found the average time it took to resolve a ticket for each course. Then for every ticket that a tutor resolved I divided the time it took them to resolve that ticket by the average for that course. For each tutor I took the average across all their tickets and this became their tutor bias term. If a tutor was particularly speedy they might have a tutor bias term of .5, but if they tend to take their time with each student they might have a tutor bias of 1.7. I threw out any ticket that took less then 5 minutes or more than an hour to prevent outliers from skewing the data. To calculate the tutor bias term for a ticket, I took the average tutor bias of all the tutors on duty.

Average Time to Resolve the Last 10 Tickets –
Certain assignments take more time to tutor then others, but I had no way of knowing which assignment the ticket was for. Instead I figured that using the resolution time of the last 10 tickets would be a good catch all for all the features I wasn't able to include in the model. I calculated the average time to resolve the last 10 tickets by traversing the history of ticket events and keeping a linked list of size 10 for each course. If a ticket took less than 5 minutes or more than an hour to resolve I discarded it as an outlier.

III. DESCIBE MODELS

Naïve Baseline –
The naïve baseline only has one parameter. It makes use of none of the information unique to a ticket. I used the numpy linear algebra least squares function to set the single parameter, and it set it to around the mean wait time of all tickets.

Single Layer Perceptron –

To make the single layer perceptron useful I had to determine how the features should be related myself. This became a very involved process. I started out by taking the position and dividing it by the number of tutors, I then multiplied that term by the tutor bias and the average of the last 10 tickets. I could have made all these feature their own linear feature, but I multiplying them increased the accuracy of the model. If you are fifth in line and there are five tutors on duty it should take about the same amount of time as if you were first in line with only one tutor. On the other hand, five tutors on duty at the same time may work more than five times faster than a single tutor because they have more shared knowledge. So I added another term that is just the number of tutors on duty. Of course the gains of having multiple tutors on duty at the same time is probably not linear, so I had to try and figure out how I should break down the one hot encoding. I did a similar process for adding a standalone feature for position, with the assumption that is you are position 75 on the queue the equation for predicting wait time probably changes. Making these decisions was not easy. If someone else had done this same assignment they would most likely made some different decisions. In the end my model had 31 features.
(position * tutor bias * avg last 10 tickets / num of tutors) one for each of the 5 courses with more than 5,000 tickets and then one as the base, (position on queue) one for each course and a base course, (position on queue if the position if greater then 25) one for each course and a base course, (number of tutors) one for each course and a base course, (number of tutors if the number of tutors is greater than 5) one for each course and a base course, (intercept term that is always 1). I trained the model using the numpy linear algebra least square method.

Multilevel Perceptron with 31 features –

A multilevel perceptron will determine the appropriate relationship between features through backpropagation so there was no need for me too multiple or divide any two features. However, because the multilayer perceptron has to use linear regression instead of solving the equation outright using linear algebra, I z-scored all the features. This way the regularizer did not penalize features with larger scales. I wanted to compare the performance of the multi-level perceptron with the single level perceptron so I kept the number of parameters between the two as close as possible. The base layer has 11 features (position, avg last 10 tickets, number tutors, tutor bias, intercept term, and one hot encoding of the courses). The second and third layer have 2 terms each, and the forth layer just has a single layer. All of the hidden layers have an activation function of tanh. I used a learning rate of 0.001, a weight decay term of 0.005, and the stochastic gradient descent optimizer function using mean squared error as the loss function. I trained it for 20 epochs on half of the total data.

Multilevel Perceptron with 221 features –

After experimenting around with different architectures for my multilevel perceptron I got by best results with a perceptron that had the same 11 base features, then two hidden layers each with 10 parameters, and then an output layer with a single parameter. I also reduced my learning rate to 0.0005 and increased my regularizer to 0.007.

Comparing the Models –

The performance of the single layer perceptron relied heavily on the choice made by the architect on how the features should be represented. Making these decisions took a considerable amount of time and my ability to infer the underlying functions was unreliable. A major advantage of the multilevel perceptron was its ability to learn these dependencies on its own. The multilevel perceptron had its own challenges though. Setting the learning rate, the regularization term, and the architecture of the perceptron took experimentation to get right and each experiment takes a couple of minutes to run. I also had to use a validation set to prevent over fitting. All things considered, the multilevel perceptron took less time to refine and got better results.

IV. LITERATURE

How Have Others Solved Similar Problems –

Before starting on my project I identified two similar prediction task and looked to them for some guidance. The first task I looked at was google maps' estimated arrival time feature. In particular I wanted to how much google relies on recent travels times versus historic travel times. Unfortunately, google still finds the inner workings of maps to be too valuable to share with the public, but I did find a couple of articles describing the difficulties that the maps team face when making their predictions. Traffic is inherently unpredictable! Without omnipotent knowledge, google cannot predict a car crash 5 hours from now, or how a rain storm will affect a backcountry highway. This gave me some reassurance when my own predictor did preform as well as I was hoping. One of the best ways to improve your predictor is to increase the amount of information it knows.

The second analogous task I researched was predicting how long a process has to wait for a resource for operating system optimizations. The biggest take away I got from that paper was the importance of removing outliers from my data. This lead me to read other papers about the effects of outliers and in the end removing outliers noticeably increased the accuracy of my predictor.

Correlation Is Not Causation –

I struggled conceptually for quite a while over whether I should keep day of the week and time of day as features in my model. Assume for the sake of argument that the tutor bias term and the day of the week are 100% correlated. Tutors who take their time always tutor on Tuesdays, Sundays, and Thursdays, and speedy tutors always tutor on the other days. Are there negative effects of keeping day of the week as a feature? I decided to consult the literature. In the *Dangers of Data Mining* Sullivan, Timmermann, and White talk about a similar situation in the field of finance. As early as the 1930 academics have published papers reporting on calendar effects on the stock market [3]. The studies describe anomalies which if acted upon would make a potential investor very wealthy simply by buying stocks on Monday and selling them on Thursday. Sullivan et al argues that these predictors are fragile to change. If the analyst were to review data from the 80's they may find Mondays to be the best day to buy, but if they reviewed data from the 90's it might be Tuesday that is the best day to buy. With this story in mind I decided to remove day of week and time of day as features from my models. I wanted to make my models as resilient to changing data as possible.

Dealing with Outliers –

After reading Osborne and Overbay I realized how many different types of outliers could be in my dataset. I knew I had some sampling error outliers (tickets which are labeled as taking six hours, but only because the tutor forgot to hit the button to resolve the ticket). Of course there might be some sampling errors that are not outliers, too. On top of that there are some legitimate outliers in my data as well (tickets that legitimately took 2 hours to resolve). Considering the intent of my predictive task I decided to remove all my outliers. Since my data was from a non-normal distribution I found that roughly 95% of tickets take less than an hour to resolve, so I considered any ticket that took longer than an hour to resolve an outlier.

V. Results

Naïve Baseline –

Mean Squared Error = 2375907.98
Standard Deviation = 7345198.49

Single Layer Perceptron –

Mean Squared Error = 247.17
Standard Deviation = 778.09

Small Multilevel Perceptron –

Mean Squared Error = 206.97
Standard Deviation = 606.54

Large Multilevel Perceptron –

Mean Squared Error = 201.61
Standard Deviation = 588.43

Evaluating the final product –

The final product does a reasonable job a predicting wait time. 80% of tickets take within 10 minutes of their predicted time to receive help, and 95% of tickets take within 30 minutes of their predicted time to receive help. These numbers are not quite as good as I'd hoped. If I were a student using the queue (which I often am) I would not be able to depend on the wait time statistic for planning my day if I knew there was a 5% chance it'll be off by more than 30 minutes. The model makes its worst mistakes when the when it expects a ticket to take a very long time, but for whatever reason the ticket is helped much sooner than expected.

Conclusions –

I learned some important lessons from this project. First I learned that multilevel perceptrons are much more convenient than single level perceptrons. Second I was reminded that correlation is not causation. Just because the day of the week may be highly correlated with wait time does not mean that the day is what causes the wait time to be lower on Monday and higher on Sunday. Third, outliers can ruin a perfectly good mean especially when your data is prone to miss labeling. Forth, the recent past is a good predictor of the recent future.

References

[1] Smith W., Taylor V., Foster I. (1999) Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In: Feitelson D.G., Rudolph L. (eds) Job Scheduling Strategies for Parallel Processing. JSSPP 1999. Lecture Notes in Computer Science, vol 1659. Springer, Berlin, Heidelberg

[2] Epstein, Zach. "Former Google Engineer Explains How Google Maps Determines Your ETA". *BGR*. N.p., 2017. Web. 13 Mar. 2017.

[3] Ryan Sullivan, Allan Timmermann, Halbert White, Dangers of data mining: The case of calendar effects in stock returns, Journal of Econometrics, Volume 105, Issue 1, November 2001, Pages 249-286, ISSN 0304-4076, http://dx.doi.org/10.1016/S0304-4076(01)00077-X.

[4] Osborne, Jason W., and Amy Overbay. "The power of outliers (and why researchers should always check for them)." *Practical assessment, research & evaluation* 9.6 (2004): 1-12.

[5] Selst, Mark Van, and Pierre Jolicoeur. "A solution to the effect of sample size on outlier elimination." *The quarterly journal of experimental psychology* 47.3 (1994): 631-650.