Predicting CS106 Office Hours Queuing Times

Nick Troccoli
troccoli@stanford.edu

Brahm Capoor
brahm@stanford.edu

Michael Troute
mtroute@stanford.edu

Abstract

We apply machine learning to predict the time a student at CS106 office hours will wait before being helped, and the time a member of the course staff will take to resolve that student's problem. We frame the prediction task both as a regression problem (applying neural networks) and a classification problem (applying logistic regression and neural networks). For classification, neural networks and logistic regression act very comparably; for regression, a neural network is able to predict wait times with a mean squared error (MSE) of 1756 *and help times with an MSE of* 387. *Overall, classification methods provide the best results, accurately classifying up to 42% of wait times and 18% of help times within 5 minute buckets, and up to 73% of wait times and 59% of help times within 20 minute buckets.*

1. Introduction

Each year, thousands of students enroll in Stanford's introductory CS106 computer science classes (CS106A, CS106B and CS106X). As such, wait times at office hours (the "LaIR") can be significant. These wait times can be impacted by a number of factors, such as the time the request is made, which helpers are currently on duty, and the students history at the LaIR, among many others. As a student, it can be difficult to plan work around office hours; as a member of course staff, on busy nights it can be difficult to gauge when to no longer accept signups so that all outstanding requests will be addressed before closing. Thus, accurately estimating wait and help time is important because students can better plan their time, and course staff can know in advance the time required to resolve outstanding requests. Further, analyzing the trained model could provide insights into what factors most impact help times.

Our goal is to compare the performance of classification and regression models to examine whether we achieve higher accuracy by predicting exact wait and help times, or by approximating these times within certain "time buckets" of varying sizes. We also aimed to generate separate models for wait and help times with the expectation that they would be influenced by different features. Ultimately, we hope our work will help the CS106 classes better understand and manage help resources for their students.

2. Related Work

Existing work in predicting queueing and service times have formulated the prediction task differently. Some existing approaches focus on first predicting service times for existing requests in the queue, and then running probabilistic simulations to reverse-engineer wait-time[4]. Other approaches use regression techniques such as support vector regression to predict the probability density functions for request arrivals and request service times, and using those densities to estimate predicted queue load at a future point in time[3]. The attempts that have been made to approach the prediction of queueing times directly as a machine learning problem have focused on predicting wait times (only, excluding service times) over a number of different locations, with some form of load balancing as the ultimate goal[5]; a notable exception is work that has been done in predicting patient wait times for scheduled and unscheduled medical procedures, which achieved a fair degree of success[2]. Our approach is distinct from existing work in that we approach the prediction of wait (queue) and help (service) times directly as machine learning tasks, rather than simulation tasks aided by learning, and we aim to generate predictions for a single queue for informational purposes, rather than for many queues for load balancing purposes. We feel that we can approach queuing times without relying on simulation due to the size and quality of our dataset, to be discussed in the next section.

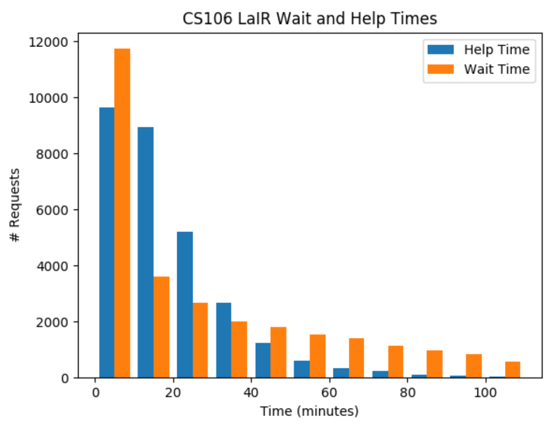
3. Dataset and Features

Our dataset consist of all LaIR help requests made in the past 8 quarters, for a total of 29,357 requests. We can obtain this information since LaIR signups are made on a designated LaIR computer, which stores information about each request in a custom database. Each LaIR request record includes the following information:

1. An anonymized, unique identifier for the requesting student

2. The course (CS106A, B, or X) in which the requesting student is currently enrolled
3. The academic quarter (Autumn, Winter, or Spring) in which the request was made
4. A description, submitted by the student, of the problem or question they have
5. A list of anonymous, unique identifiers for each course staff member working at the LaIR when the request was made
6. A timestamp for when the request was made
7. A timestamp for when the request was claimed by a member of the course staff
8. A timestamp for when the request was marked as resolved by the course staff member

However, we did have to perform some data cleansing due to some inaccurate data. The first problem we encountered was, due to high wait times at the LaIR, it is sometimes the case that a student will leave before they are helped, resulting in a wait time that simply represents a staff member resolving the request. Second, course helpers sometimes forget to mark requests as resolved, leaving them open for many hours after the request has actually been completed. In both of these cases, while the wait time for the request is accurate, the help time is innacurate. To handle this, we perform some general filtering of our entire dataset, removing requests that, for instance, were closed after 3AM the day after the request was made. Moreover, when training our help time models, we only used requests with a help time of at least 2 minutes, as we deemed help times less than two minutes a result of students leaving. You can see the distribution of our dataset below, which includes those with help times under two minutes.



You also notice the extreme skew in our dataset, for both wait and help times. We consider this skew in training our models, discussed in a future section.

4. Methods

4.1. Vectorizers for Feature Extraction

We use vectorizers to generate numerical features for each LaIR request before training and evaluation. We chose featurizers and extractors for information about the request and LaIR state, explained below, that we felt were strong indicators of wait and help times. These featurizers operate on our dataset before being inputted into our model.

4.1.1 TF-IDF Request Text Vectorizer

A student's description of their problem has the potential to be a powerful indicator of wait and help time. For that reason, we used TF-IDF (term frequency-inverse document frequency) to numerically represent each request's description. TF-IDF[1] is a measure of the relevance and significance of each word in the LaIR request descriptions. It assigns each word a weight that is higher if the word appears frequently in fewer documents, or rarely in many documents. Its weight is lower if the word appears frequently in many documents. We chose this method because it allowed us to reduce noise from various prepositions (”a”, ”the”, etc.) in request text and instead focus on potentially significant words (such as ”bug”, or the name of a given assignment). We also chose this instead of using word embeddings such as GLoVe vectors because we found that LaIR requests contain a variety of domain-specific language, such as assignment names, Java terminology, and other words not commonly used in speech or writing.

One note of interest is we noticed that many students put nondescriptive descriptions when signing up for help. For this reason, we aggressively added additional features, detailed below, to predict help and wait times.

4.1.2 Helper ID Vectorizer

We imagined that the number and characteristics of course staff on duty have an impact on how a request is handled. For this reason, we created a Helper ID Vectorizer, which creates an indicator vector for all the course staff on duty at the LaIR when a request is made. Each element in the produced vector represents the presence (1) or absence (0) of one of the course staff in our dataset.

4.1.3 Course ID Vectorizer

We also believed that a student's course may provide wait and help time insight. Thus, the Course ID Vectorizer creates a an indicator vector representing the course to which a given request belongs (e.g. [1, 0, 0] for CS106A, or [0, 1, 0] for CS106B).

4.1.4 Past Requests Vectorizer

On a given night, the LaIR may experience extreme delays, depending on how many other students are there. For this reason, we vectorized certain information about the state of the LaIR at the time of the request, including wait time and help time that day, as well as the number of currently enqueued requests.

4.1.5 Student Vectorizer

A student's past history in the LaIR may also provide insight into their future experiences. For this reason, the Student Vectorizer encapsulates information about the requesting student's history at the LaIR. This includes the total amount of time this student has spent being helped at the LaIR (prior to making the request in question), and the total number of help requests they have made.

4.1.6 Request Time Vectorizer

The time of the request may also impact how quickly a student's request is claimed and resolved. For this reason, we vectorized information about the time a request is made, including an indicator representation of the quarter, the day of the week, and how far into LaIR hours the request is made.

4.1.7 Due Date Vectorizer

Course staff also experience extremely high numbers of requests surrounding assignment due dates for the various CS106 classes. For this reason, we created a Due Date Vectorizer, which includes the number of days until the next CS106A, CS106B, and CS106X due dates, respectively.

4.2. Regression Models

With our dataset successfully vectorized, we move to our first modeling of this problem: we seek to predict, via two separate regression tasks: precisely how long a student will have to wait to be helped, and once their request is claimed by a member of course staff, how long it will take them to be helped.

4.2.1 Training Set

For these two models, we produce two training sets consisting of $< x, y >$ pairs such that each x is the feature vector produced by our vectorizers for a particular request, and y is how many minutes a student had to wait, and how many minutes the student took to be helped, respectively. Our training sets consist of 60% of the requests in our dataset, or approximately 16,000 requests. The remaining requests are split into development and test sets.

4.2.2 Models

We train and evaluate using both Linear Regression and Neural Network models. The Neural Networks are each comprised of 5 size-1000 hidden layers with ELU activations, trained with Stochastic Gradient Descent (SGD) and a batch size of 25. Their loss function is Mean Squared Error (MSE). The Linear Regression models are trained using SGD as well.

4.3. Classification Models

Next, we frame the problem as a classification problem; that is, we seek to train two separate models to place a students wait time and help time into discrete N-minute buckets (for example, placing a students wait time into 5 minute buckets (e.g. 0-5 minutes, 5-10 minutes...) and their help time into 10 minute buckets.

4.3.1 Training Set

For these two models, our procedure for generating training sets is largely the same as that for regression. The main difference is, instead of labels being the exact wait or help time, they are the bucket into which the request lies. For instance, using 10 minute buckets for both wait and help time, if a students request takes 26 minutes to be claimed and 17 minutes to be resolved, their wait time is bucketed into the 20-30 minute bucket and their help time into the 10-20 minute bucket. We use 0-indexed numbers to represent these buckets as labels (e.g. 0 for the 0-10 minute bucket), and label all requests with a time greater than 2 hours (what we chose as a reasonable ”maximum” time) in the highest-indexed bucket.

4.3.2 Models

We train and evaluate using both Logistic Regression and Neural Network models. Our Neural Network architecture is identical to that for regression, except our output layer uses Softmax and Cross Entropy loss to output a predicted time bucket.

5. Experiments

5.1. Linear Regression

Running standard linear regression on our dataset until convergence, we achieved the following MSE:

Linear Regression MSE

	Dev	Test
Help Time	26536348436.13934	26795466658.66747
Wait Time	1352831512.280824	1362499621.0853095

While these results are certainly poor, they do reinforce the fact that our complex featurizers produce a resulting dataset that is far from linearly related. It is worth noting, however, the drastic difference in performance between wait and help times. In particular, the test performance shows the wait time model is almost 20x more accurate than the help time model, suggesting that wait time is almost certainly the easier of the two metrics to predict.

5.2. Neural Regression

Next, we ran our neural network regression models, in search of better results. Before doing this, we ran hyperparameter optimization for 10 epochs for both the wait time model and help time model to find an optimal learning rate (LR) with which to train. We found that the optimal learning rate for Help Time Regression was 0.02, and the optimal learning reate for Wait time was 0.01. With these learning rates, we then trained for 50 epochs and evaluated our models, obtaining the following results:

Neural Regression MSE

	Train	Dev	Test
Help Time	377.27121	411.104736	389.46875
Wait Time	1460.849609	1433.941528	1351.940674

As seen above, neural regression performs significantly better than linear regression; however, it still performs poorly. Interestingly enough, wait time has worse results than help time, while for linear regression it was the inverse. We were unable to explain this discrepency; however, as both errors are extremely high, a more accurate regression model may have drastically different performance for wait and help time.

5.3. Logistic Regression

Running standard logistic regression on our dataset until convergence (bucketing times up to 120 minutes), we achieved the following results:

Logistic Regression - Help Time

	Train	Dev	Test
# Buckets	Acc. (%)	Acc. (%)	Acc. (%)
24	19.5023	18.1682	18.1954
12	34.0835	33.033	33.4519
8	45.7435	43.8063	44.4403
6	59.8671	58.5773	58.8918

Logistic Regression - Wait Time

	Train	Dev	Test
# Buckets	Acc. (%)	Acc. (%)	Acc. (%)
24	42.699	39.6355	42.1492
12	56.4835	54.6074	56.5395
8	65.1981	64.4354	65.0034
6	72.1074	71.2144	72.4285

As you can see, these are strong results that indicate the potential to predict, for instance, wait times within 20 minute buckets with over 70% accuracy.

5.4. Neural Classification

After training our wait time neural network for 100 epochs and our help time network for 50 epochs, we achieved the following results. Note that this is with our optimized learning rates of 0.0001 for help time, and 0.00001 for wait time, determined via hyperparameter search over 10 epochs:

Neural Network - Help Time

	Train	Dev	Test
# Buckets	Acc. (%)	Acc. (%)	Acc. (%)
24	13.96	15.24	15.84
12	32.77	32.66	33.68
8	42.01	43.15	39.12
6	58.1	57.92	59.06

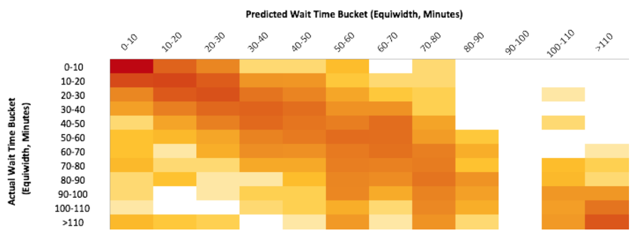
Neural Network - Wait Time

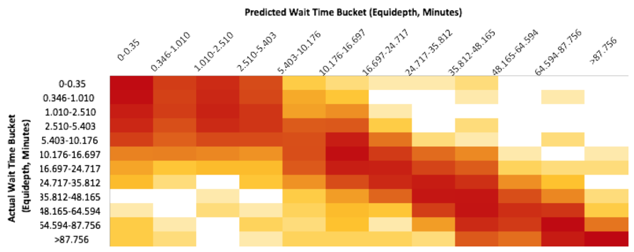
	Train	Dev	Test
# Buckets	Acc. (%)	Acc. (%)	Acc. (%)
24	14.43	35.94	37.8
12	21.6	53.86	56.16
8	27.88	64.49	65.92
6	32.53	71.9	73.3

As you can see, these are similar (but in some cases slightly better) results than those of logistic regression.

5.5. Bucketing

Due to the high concentration of examples with low wait and help times, our data is skewed towards lower buckets (in the classification setting). To combat this skew, we attempt using equidepth buckets (as opposed to traditional equiwidth bucketing). Unlike equiwidth buckets, which are buckets representing equal amounts of time (e.g. 5 minute buckets), Equidepth buckets are defined so that the same number of examples in the training set are in each bucket; therefore, the width of each equidepth bucket is variable (e.g. 0-5 minutes, 5-6 minutes, etc.). We were curious to see if this approach would lead to more accurate predictions. The following two confusion matrices were generated by training and testing a model on equiwidth and equidepth buckets, respectively:





The strong diagonal in the equidepth image represents the improved misclassification accuracy of the equidepth model; that is, when the equidepth model predicts the incorrect bucket, it tends to be closer to ground truth than the equiwidth model. The following table demonstrates these results more concretely:

	Equiwidth, 12 Buckets	Equidepth, 12 Buckets
Distance From Correct Bucket	% of Classifications	% of Classifications
0 (Correct)	56.44	31.44
1	25.44	39.53
2	9.99	17.90
3	4.32	7.00
4	1.94	2.40
5	0.59	0.77
> 5	1.28	0.97

As shown above, the equidepth buckets classify about the same percentage of results correctly within 3 buckets, but then have more accurate results for those misclassified by more than 3 buckets. Unfortunately, however, when used with logistic and neural classification, this experiment did not ultimately yield strong results.

Logistic Regression - Help Time (Equidepth)

# Buckets	Train Acc. (%)	Dev Acc. (%)	Test Acc. (%)
24	6.8205	5.1426	4.9607
12	11.6161	9.3844	10.614

Logistic Regression - Wait Time (Equidepth)

# Buckets	Train Acc. (%)	Dev Acc. (%)	Test Acc. (%)
24	23.294	19.9285	19.4482
12	38.8044	35.3262	35.5416

Neural Classification - Help Time (Equidepth)

# Buckets	Train Acc. (%)	Dev Acc. (%)	Test Acc. (%)
24	3.7725	3.9602	4.137
12	8.3709	8.6712	8.48

Neural Classification - Wait Time (Equidepth)

# Buckets	Train Acc. (%)	Dev Acc. (%)	Test Acc. (%)
24	4.1049	17.1351	16.7745
12	8.0225	31.7493	31.4373

We believe these lower results are the result of the equidepth buckets being sometimes unpredictably small in the times it represents, and also reliant on the training set to determine these buckets. However, this experiment did increase our interest in experimenting with other bucketing techniques, such as logarithmic bucket sizes, in the future, that are not simply modeled off the training set.

6. Comparison and Discussion

As shown above, our classification model clearly outperforms our regression model. Overall, classification models outperform regression models in predicting both wait and help times. The regressive neural networks for wait and help times predict, on average, help times within +/- 20 minutes and wait times within +/- 42 minutes, while the classifying network is able accurately classify 60% of requests into 20-minute buckets for help time and 65% of of requests into 20-minute buckets for wait time. Predictably, as the number of buckets decreases, our classification accuracy increases, ultimately to over 70% with wait time and buckets of 20 minutes. As predicted, our model is more successful at predicting wait time than help time, likely because of the difficulty in accurately assessing the difficulty of a student's question vs. the relative load on the LaIR at a given time. Finally, we note that our neural network classification model seems to perform similarly to logistic regression, suggesting that additional layers and non-linearities in the neural network do not significantly improve accuracy. This may indicate that, without better features, we may not be able to perform significantly better with our current model.

7. Conclusion and Future Work

We thus conclude that classification is a more promising modeling technique for this problem than regression. This solution is still suitable for LaIR use, as students oftentimes do not need a precise-within-the-minute prediction of their help or wait time. In future experiments, we hope to explore the use of custom-trained word embeddings for request descriptions, as well as additional featurizers and types of models, with the hopes of ultimately deploying these classification models for production use in the LaIR.

8. Contributions

All three teammates made significant contributions to this project. Code at https://github.com/Nickster28/CS229-Aut17-Project.

References

[1] Tf-idf weighting at https://nlp.stanford.edu/ir-book/html/htmledition/tf-idf-weighting-1.html.

[2] C. Curtis, C. Liu, T. J. Bollerman, and O. S. Pianykh. Machine learning for predicting patient wait times and appointment delays. *Journal of the American College of Radiology*, 2017.

[3] G. sheng Hu and F. qi Deng. The analysis of queuing system based on support vector machine. 3:2320–2325 Vol. 3, Dec 2004.

[4] I. F. Warren Smith, Valerie Taylor. Using run-time predictions to estimate queue wait times and improve scheduler performance.

[5] J. Z. Ye Zhang, Le T. Nguyen. Wait time prediction: How to avoid waiting in lines?