An Exploration of the COS 126/226/217 Lab Queue

Grace Guan (`gg17`)

Every semester, over 500 students enroll in Princeton's introductory computer science classes COS 126 (Computer Science: An Interdisciplinary Approach), COS 226 (Algorithms and Data Structures), and COS 217 (Introduction to Programming Systems). To assist students currently in the courses with debugging code, the COS department hires Lab TAs, who are undergraduate students who have successfully completed these three courses. To manage demand for these Lab TAs, students place their name on a Lab Queue, and Lab TAs address student requests in a first-come-first-served fashion. The Lab Queue is currently hosted at `https://www.labqueue.io/`*. The queue faces high demand at certain peak hours, specifically the day before major assignments (e.g. N-Body, Othello) are due, and the current wait time predictions of the queue are not accurate. The lack of predictability in wait times of the Lab Queue both drives students away from and lowers ratings of COS classes at Princeton.*

Through this exploratory research project, I provide the following deliverables: (1) A Jupyter Notebook that creates a cleaned-up, normalized, anonymized dataset of the Lab Queue all terms available (Fall 2017-Fall 2019) to be useful for future ML analyses and suitable for public release; (2) An improved framework for the wait time estimator that is possible to integrate into the Lab Queue v3 code currently deployed online.

Contents

1 Dataset Creation and Documentation

The first two work orders were as follows:

1. Clean-up/normalize the dataset of the Lab Queue for all terms available (up to Fall 2019) to make it ready to use for ML analyses (by myself and others)

2. Anonymization of the dataset (removal of NetIDs) such that it is suitable for public release

I created two Jupyter Notebooks written in Python 3.7.6 to complete these tasks. The goal of these notebooks is to be written in a documented and easily understandable manner so that the work can be extended by others.

First, the Jupyter Notebook `clean_anonymize_basicplot_data.ipynb` takes in a json file with raw data from the lab queue and outputs a cleaned up and anonymized CSV file named `labqueue_anon.csv`. Then, I further clean up the outputted file in `remove_outliers.ipynb`, by cautiously deleting outliers[1] from the dataset. The outlier removal notebook outputs a final file named `labqueue_anon_1.csv`.

Each row in `labqueue_anon_1.csv` represents one student request in the lab queue. The columns in `labqueue_anon_1.csv` include:

1. `course` - The course the request was made for. Possible values include COS 109, 126, 226, and 217.

2. `description` - The description of the request, in text.

3. `is_open` - Whether the request is open, a boolean.

4. `location` - Location of the request in text, most of the times Lewis 121 or 122. Sometimes "unsure" if the student didn't know which room they were located in. Entries before February 13, 2018 are missing this value.

5. `pk` - Unique integer identification value for this request. Strictly monotone increasing.

6. `time_accepted` - Time the Lab TA started helping the student, string when the dataframe is read in from CSV, but methods are provided to turn this into a Timestamp data type.

[1] We defined outliers as `time_accepted==time_closed` and `time_created` came before the lab queue hours started (7 pm for weekdays, 3 pm for Saturdays, and 5 pm for Sundays). Additionally, rows corresponding to certain `pk` values were deleted after features were generated because their wait times were more than double any other wait time from that day, which could potentially correspond to students "abusing" the queue. Lastly, all requests made by Maia Ginsburg were assumed to be test requests and were deleted as outliers.

7. `time_closed` - Time the Lab TA finished helping the student, string when the dataframe is read in from CSV, but methods are provided to turn this into a Timestamp data type. Sometimes, Lab TAs forget to close requests, so there is some amount of noise in this feature.

8. `time_created` - Time the student put their name on the Lab Queue, string when the dataframe is read in from CSV, but methods are provided to turn this into a Timestamp data type.

9. `wait_time` - Difference between `time_accepted` and `time_created`, a Timedelta data type.

10. `help_time` - Difference between `time_closed` and `time_accepted`, a Timedelta data type.

11. `wait_time_in_minutes` - Difference between `time_accepted` and `time_created`, a float64.

12. `help_time_in_minutes` - Difference between `time_closed` and `time_accepted`, a float64.

13. `TA_anon` - Unique integer identifier for the TA handling the request, generated at random (no way to back-link id to TA).

14. `student_anon` - Unique integer identifier for the student submitting the request, generated at random (no way to back-link id to student).

15. `day_created`, `hour_created`, `minute_created` - The day, hour, and minute the request was created, all integers.

16. `year`, `month`, `day`, `hour_accepted`, `minute_accepted` - The year, month, day, hour, and minute the request was accepted, all integers.

17. `hour_closed`, `minute_closed` - The hour and minute the request was closed, all integers.

18. `day_of_week` - The integer day of week of the request, with Monday as 0 and Sunday as 6.

19. `week_of_year` - The integer week of year of the request, using the isocalendar[2].

[2]`https://docs.python.org/2/library/datetime.html#datetime.date.isocalendar`

20. `Color`, `Marker` - Color and marker for plotting. Monday is red, Tuesday is chocolate, Wednesday is gold, Thursday is green, Friday is cyan, Saturday is blue, and Sunday is magenta. COS 109 is a $\blacklozenge$, COS 126 is a $\cdot$, COS 226 is a $+$, and COS 217 is a $\times$.

21. `cos109`, `cos126`, `cos217`, `cos226` - One-hot variables (integer 0 or 1) representing whether the request was for that class.

22. `isMonday`, `isTuesday`, `isWednesday`, `isThursday`, `isFriday`, `isSaturday`, `isSunday` - One-hot variables representing whether the request was made on that day.

23. `woyn` where $n \in \{2-3, 6-20, 37-50\}$ - One-hot variables (integer 0 or 1) representing whether the request was made on that week of year. Correspondingly, `sem_week_m` where $m \in \{1, 14\}$ - One-hot variables (integer 0 or 1) representing wehther the request was made on that week of the semester. The mapping of week to school year is difficult because fall semester currently has an extra week due to Thanksgiving. The fall semester weeks correspond to calendar weeks $37-50$, 2, and 3 (for Dean's Date work in January), whereas the spring semester weeks correspond to calendar weeks $6-20$. Additionally, Thanksgiving doesn't fall on the same week every year, so this task becomes a little tricky. I decided to lump the first and second weeks of the school year together in the fall semester (because there are 7 weeks before break in the fall), and then have all following weeks be as numerically ordered. I looked up the break schedules for the past 4 semesters, and saw that in Fall 2017 and 2018, Fall break was the 44th week of the year; Thanksgiving break was the 47th week of the year. In Spring 2018 and 2019, Spring break fell on the 12th week of the year. **In future work, custom functions that map `woyn` to `sem_week_m` should be carefully defined because of academic year calendar changes.**

24. `wait_time_90m_before_submit`, `wait_time_30m_before_submit` - Mean wait time for all requests accepted within in the 90 and 30 minutes before the request was created, respectively, in Float64.

25. `help_time_90m_before_submit`, `help_time_30m_before_submit` - Mean help time for all requests closed within in the 90 and 30 minutes before the request was created, respectively, in Float64.

26. `num_in_queue_before_us` - How many other requests had a creation time before this request's creation time, but a closed time after this request's creation time (although this does introduce some noise because of the noise in closing requests). Integer.

27. `time_created_floor` - Timestamp data type floor of the time created by hour. For example, a request created at 17:48 would have a floor of 17:00.

28. `TAs_this_hour` - Integer number of unique TAs on staff at the time the request was created for the hour beginning at the floor of this request's `time_created`. However, this does not take into account potential TAs who could have stayed longer into another shift, or students whose wait time spans multiple hours with different numbers of TAs.

29. `day_before_126_due`, `day_of_126_due`, `day_before_226_due`, `day_of_226_due`, `day_before_217_due`, `day_of_217_due` - One-hot variables (integer 0 or 1) representing whether the request was made the day before or of a COS 126/226/217 assignment was due. Does not take into consideration what class the request was made for.

30. `assignment_due_next_day`, `assignment_due_day_of` - One-hot variables (integer 0 or 1) representing whether the student who submitted the request had their assignment due the next day or the day of.

31. `more_TAs_than_queue` - One-hot variable (integer 0 or 1) representing whether the number of TAs is $\geq$ the number of students in the queue prior to this student.

Many (though not all) of these features would be useful in ML analyses of Lab Queue data. Additionally, by nature of not containing NetIDs, this dataset is fully anonymized because `TA_anon` and `student_anon` cannot be traced back to the NetIDs that originally generated them. Thus, the objectives of the first two work orders are complete.

2 Data Insights

Figures 1 and 2 present the volume of each day as well as the wait times for each day for a typical span of two weeks for the Lab Queue and the week leading up to Dean's Date in the Fall 2017 semester respectively. The x-axis does not correspond to any linear timeframe. We separate data points by weekday (color) and by class (shape). On "peak" days, the manner in which the wait time increases throughout the shift and never lowers down indicates that a lot of students may have went home without their questions answered. Wait times on a "peak" day often exceed 60 minutes in the later hours of a shift, compared to wait times on the day after major assignments are due which rarely exceed 20 minutes (Compare Figure 3 to Figure 4).

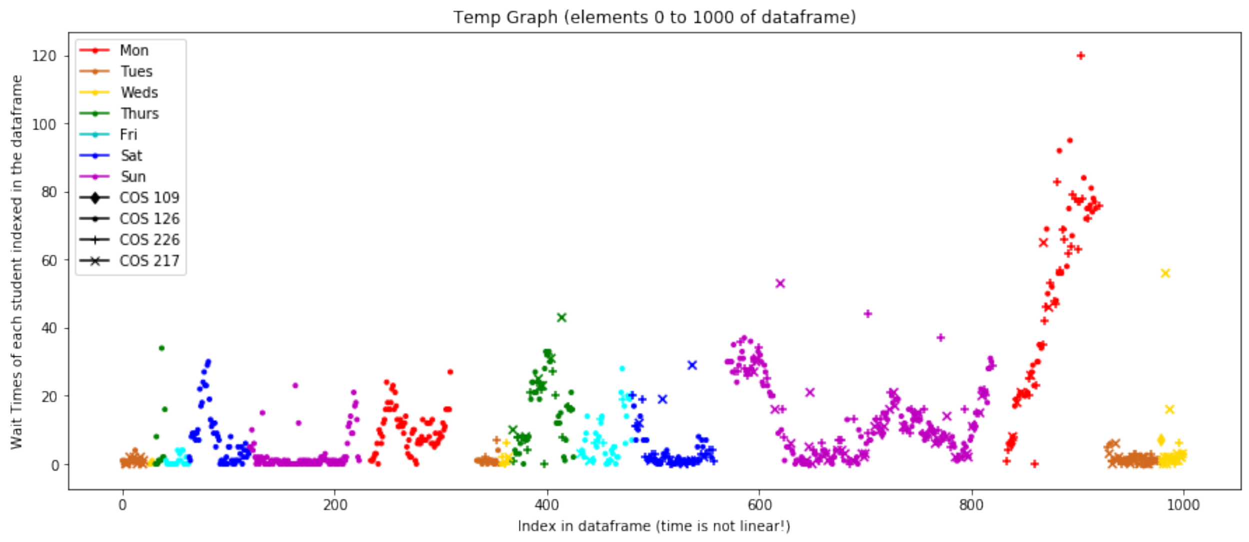

Figure 1: Generally, wait times remain under 20 minutes in a typical two week span of the lab queue. The spike in red is the Monday that WordNet was due for COS 226, and Guitar hero was due for COS 126. This is confirmed by the presence of lots of +'s (COS 226) and ·'s (COS 126).

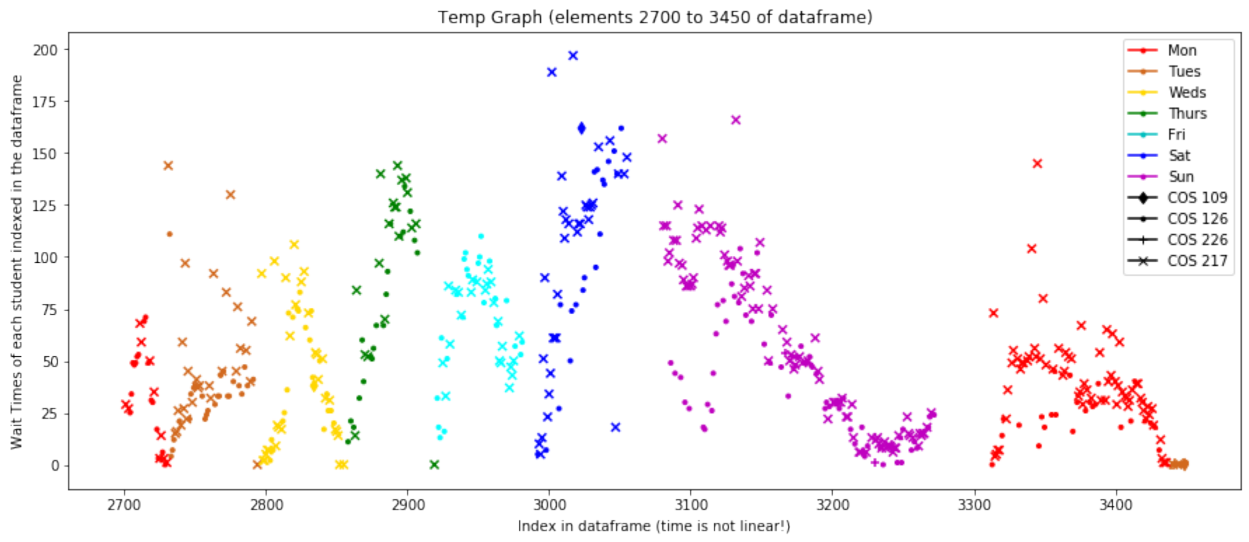

Figure 2: This figure presents wait times in the Lab Queue for January 2018, specifically the week leading up to Dean's Date for the Fall 2017 semester. The orange, yellow, and green on the right-hand side represent the first three days of the Spring 2018 semester. For all shifts in the week leading up to Dean's Date, wait times are low at the beginning of each shift and then quickly spike. Most requests appear to be coming from COS 217, which had a very difficult final project compared to the COS 126 final project. There are clusters of ·'s below the clusters of $\times$'s on the Sunday and Monday before Dean's Date, which indicates that TAs are taking COS 126 (·) students off the queue before 217 students ($\times$).

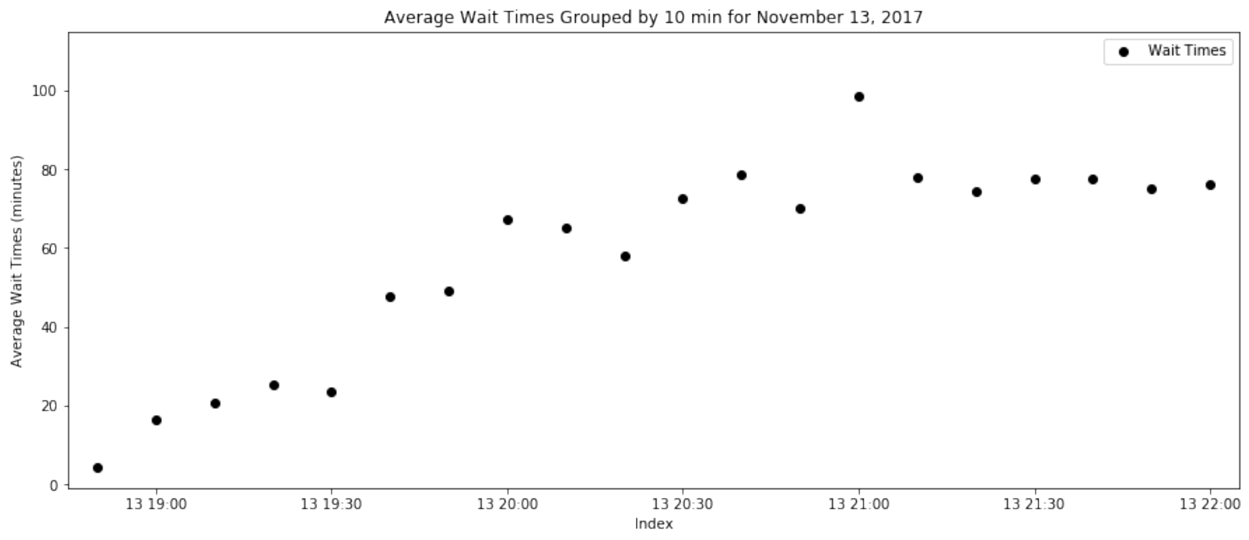

Figure 3: This figure presents the average wait times every 10 minutes for a sample "peak" day, when the Lab Queue is overwhelmed. This date, November 13, 2017, was the day that WordNet was due for COS 226 and GuitarHero was due for COS 126. The average wait time starts off around 20 minutes for the first 30 minutes of the shift, then increases to above 60 minutes after 8:00 pm.

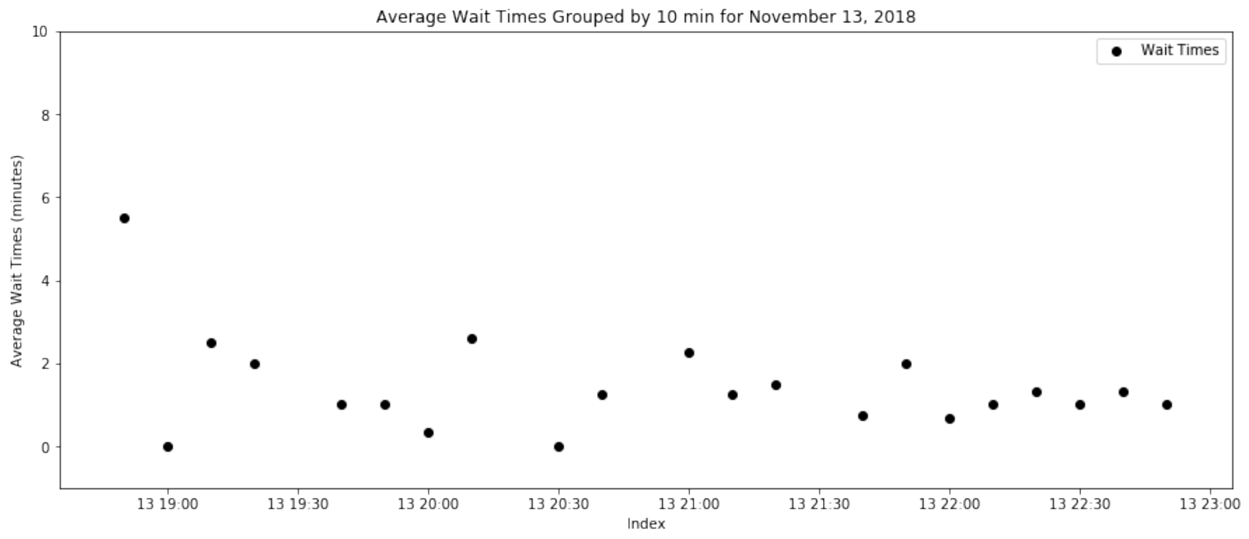

Figure 4: This figure presents the average wait times every 10 minutes for a "normal" day, when the Lab Queue has enough staff on hand to debug student requests. This date, November 13, 2018, was the day after WordNet was due for COS 226 and GuitarHero was due for COS 126. The average wait time never goes above 5 minutes. Requests come from students in COS 217 or students taking late days on COS 126 and 226 assignments.

3 New Wait Time Estimator Framework

The third work order was to be “Applying existing expertise from previous internship to see if the wait time estimator can be improved.” To do so, I built simple linear models that could potentially be run in the lab queue Python scripts themselves, and performed simple feature selection to determine the optimal model. I trained the models on the Fall 2017 (incomplete data due to migration to Lab Queue v3 in the middle of the semester), Spring 2018, and Fall 2018. I tested the models on held-out Spring 2019 data. The metrics that I used for evaluating our model performance on the test dataset are mean squared error (MSE; penalizes large outliers more heavily) and mean absolute error (MAE; easier to interpret).

I tune our model somewhat to the test dataset because the real life implementation (unseen data) is our real test dataset. Thus, our current test dataset is more of a toy validation set.

As a baseline, I used a simple linear regression model with the features sem_week_m, isWeekday (one-hot-encoded), course (one-hot-encoded), wait_time_30m_before_submit, num_in_queue_before_us, TAs_this_hour, assignment_due_next_day, assignment_due_day_of, and more_TAs_than_queue. For co-linear one-hot-encoded features, I dropped one feature from each category. Choosing these features led to the best performance on our test dataset. Upon inspection of regression coefficients, num_in_queue_before_us was the largest predictor of wait time with a coefficient of 115.03 (all other coefficients had an absolute value < 50). This led me to think that we were overfitting and that we should try a regularized model to promote sparse solutions.

Therefore, I turned to the ElasticNet model, which allows for a combination of $L1$ and $L2$ norm penalization, where α is the constant that multiplies the penalty terms and the $L1\_ratio$ is the ElasticNet mixing parameter of how much the $L1$ norm is penalized compared to the $L2$ norm. Fixing α and adjusting the $L1\_ratio$, I saw best performance when $L1\_ratio = 1$, which meant that optimal performance would occur when penalizing with the $L1$ norm only. Then fixing $L1\_ratio = 1$, I tested various features on varied α, choosing the model with least MSE. The features sem_week_m, num_in_queue_before_us, TAs_this_hour, hour_created, minute_created, more_TAs_than_queue, day_before_126/226/217_due, and day_of_126/226/217_due proved to minimize MSE. In ElasticNet regression, I included all co-linear features because the model could choose which features to ignore.

Interestingly enough, the isWeekday feature worsened performance. This could be because assignments are due on different days of the week each semester (ex. De-comment for COS 217 alternates being due between Tuesday and Wednesday) and it is a more powerful predictor to know how many days away a student is from the assignment’s due date compared to what day of the week it is. Additionally, I was surprised that the wait times and help time averages from 30-90 minutes before each individual student did not help predict their wait times, as I had thought that would be the most important feature.

The comparison between the Baseline and ElasticNet model errors on the test set can be found in Table 1.

Model	MSE	MAE
Baseline	129.326	8.695
ElasticNet	108.833	7.747

Table 1: Performance comparison between our Baseline model and our ElasticNet model on Spring 2019 data.

4 Future Work

Future work on the research side of things could incorporate the following:

- Adding a linear "number of days until 126/226/217 assignment is due" feature. This would increase the information currently encoded in the one-hot day_before_126/226/217_due and day_of_126/226/217_due variables.
- Adding a feature for how difficult the current assignment is. This was a feature requested by Jeremie early on in the project, but I was never sure how to implement it due to its subjectiveness. This may already be reflected in which week of the semester it is.
- Running 126, 226, and 217 wait time predictions separately from one another. While this would technically improve prediction and could be useful if we were giving each individual student wait time predictions (rather than displaying a "wait time estimate" on the front page of the queue as currently done), I do not think this is realistic for actually implementing the queue in real time because we should be giving one wait time estimate for all classes, not each class separately.
- Running simple neural network or RNN/LSTM models that could capture time series in the data, although this may also not be realistic for actually implementing in the queue in real time due to time and CPU constraints.

More concrete future work involves ensuring the model is portable enough to be ported to (and then porting it into) the existing Lab Queue v3 code.

5 Acknowledgements

The author thanks Dr. Jeremie Lumbroso for advising this very interesting project and Yannis Karakozis for helpful comments.