WaitLess
Real-time wait time prediction for customers

David Hirschberg
Computer Science Dept.
UC Irvine
hirschbd@uci.edu

David Hess
Computer Science Dept.
UC Irvine
hessd@uci.edu

Alex Silver
Computer Science Dept.
UC Irvine
silveraa@uci.edu

Nicholas Alba
Computer Science Dept.
UC Irvine
ndalba@uci.edu

Abstract

Abstract – In an era of consumerism and instant gratifications, time is of the essence. Thus, any time waiting for one task to complete is time that could be spent on a new task. Our research aims to focus on aggregating wait time data to provide people with real-time information about how long a single task will take. In a study by Long Range Systems, LLC, a supplier of wait time "buzzers", it is found that nearly 56% of waiting times at restaurants are 20 minutes or more. [1] This does not even include time to actually eat the meal, and thus calls for some solution to be created. There are already existing applications that have attempted to solve this problem, but have only gotten as far as relaying only the current wait time to the consumer, rather than predicting wait times later in the day. [2][3] There has been some work in wait time estimation, with the state of the art being LineKing [4], but there are some drawbacks that we improve and build upon. We have created a system that collects real-time data, and relays to the consumer both the current wait time, and projected wait times throughout the entire day.
WaitLess is able to not only beat this upper limit, but able to do far better, by being able to provide mean absolute errors (MAE) of less than 1 minute in a 10-minute prediction. To reiterate, LineKing is able to provide mean absolute errors of 2-3 minutes, [4] Waitless beats it out with mean absolute errors of less than 1 minute.

1. Novelty/Motivation

1.1. Importance of Problem

In an era of consumerism and instant gratification, time is of the essence. Thus, any time waiting for one task to be complete is time that could be spent on a new task. Our research aims to focus on aggregating wait time data to provide people with real-time information about how long a single task will take, either currently, or anytime that particular day. To make things less abstract, we are focusing taking wait time data for restaurants and other food service businesses based on user data, and providing an entire user base with live waiting time data for a given business based on the collected data.

This research is inspired by the idea that if a person can make a judgement call on if they have time to wait to eat at a restaurant or not, they can develop a more efficient daily routine. In a study by Long Range Systems, LLC, a supplier of wait time "buzzers", it is found that nearly 56% of waiting times at restaurants are 20 minutes or more. [1] This does not even include time to actually eat the meal, and thus calls for some solution to be created.

There are already existing applications that attempt to solve this problem, but have only gotten as far as relaying only the current wait time to the consumer, rather than predicting wait times later in the day. For example, "What's The Wait" is an application that takes restaurant listings from Yelp and Google and asks users to submit their current wait time; similarly, Disneyland has an app that relies on users to submit their wait times for rides. [2][3] One critical point that we are looking to improve upon is the idea of the user submitting data: we want to get passive data from the users so that the aggregate data is far more saturated (allowing for better estimation of wait times), instead of relying on a user always remembering to open our application. In terms of relating to search engines, a user will be able to query data from a variety of local restaurants to see what the current wait times are, as well as what the wait times are historically at a particular time. This ties into the idea of hyperlocal search, a next generation search technique.

1.2. Justification via Beating the State of the Art

According to a well cited paper on "*How long should a customer wait for service*" [5], "Analysis of the data reveals that the "ideal" waiting time… is significantly less than the current corporate waiting time policy." The paper goes on to explain that having an ideal waiting time creates more profit as a function of customer satisfaction.

According to another well cited paper from "*Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud*" [6], There is a tradeoff between customer satisfaction and profit, and in this paper's case,

customer satisfaction has a direct correlation with amount of time waiting, even in this high speed environment of high performance distributed computing. We need to take into account of different methods of simulations and how to create a system that can be easily used in different environments, ranging from human to human wait time interaction, and computer agent to computer agent interaction.

In academics there have been some attempts at wait time estimation, whose abilities far surpass commercial attempts. LineKing is the state of the art when it comes to wait time estimation. LineKing boasts being able to provide a mean absolute error of less than 2-3 minutes when predicting 10 minutes into the future [4]. WaitLess is able to not only beat this upper limit, but able to do far better, by being able to provide mean absolute errors (MAE) of less than 1 minute in a 10-minute prediction. To reiterate, **LineKing has a MAE of 2-3 minutes, [4] Waitless beats it out with a MAE of less than 1 minute.**

Our system will produce useful data that will inform the consumer of both current wait times and projected wait times throughout the day so that the consumer can make better informed decisions of how to best manage and spend their time.

2. Technology

2.1. Data Sources

Data has to come from somewhere. We require knowledge of what time a customer entered or exited a particular location and how long they spent there on what particular day. It is important to note that not every customer must be tracked, but the more data we have, the stronger are results become. There are several avenues from which this data can be acquired.

2.1.1 Customer Manual Feedback

We could ask customers to report manually how long then spent waiting at a location. This will not be pursued as we wish to be passive.

2.1.2 RF Based

We could install or have access to an RF chip reader in every single commercial location. Customers could be tracked throughout the building using RADAR as presented as “an in-building RF-based user location and tracking system”, the biggest downside is the cost and ubiquitousness of such systems. [7]

2.1.3 Video based

We could track customers based on a video feed from every locations surveillance footage, we could capture what time they enter and exit a location purely by tracking that person in real time using a monochromatic video provided from the aforementioned video cameras. [8] We would then have the required data to leverage the outcome we desire.

2.1.4 High Precision GPS Tracking

We can track every customer’s GPS location from their smartphone. We would have access to all of their location data from which we can gather how much time was spent at various locations, including commercial locations where wait times are of interest. We would need access to everyone’s location data, via some application that the user would need to install. [9]

2.1.5 Geo Fence

We could have a geo fence as described by a geographical polygon that encompasses a location of interest, in this case a store front. When a customer enters a location as defined by the geo fence, it is taken note of. When a customer than leaves the geo fence, we know when and how long the customer spent in that particular location. This would however still require access to GPS location, but would instead only fire at threshold crossings, greatly decreasing battery drain and computational complexity. [10]

2.1.6 Geo Fence via Wi-Fi

We can go further than geo fencing, by accessing network proximity via Wi-Fi connections. The geo fence is now when the customer connects and disconnects from a Wi-Fi router at a particular location. When a particular smart device connects and then disconnects from a router that acts as the threshold of the geo fence. [10] A filter would be needed to account for brief connections and other outliers. This method does not require access to any permissions on any smart device, rather the router is all we would need access to.

We will be using a geo fence approach using Wi-Fi as our data source. This will allow us to access all the appropriate data passively without any user interaction. We would also be able to do this without installing and requiring any software on customer phones. This will allow for a more ubiquitous tracking of customers to produce the best results.

2.2. Storage

We use a standard SQL-based data store. Its purpose would be to hold the data after being processed by our system to be in the format the prediction algorithm requires.

2.3. Software

2.3.1 Web Server

The current webserver is NGINX. NGINX is known for being high-performance and very stable. More importantly, NGINX uses an asynchronous, event-driven architecture,

and as such is far more scalable than the traditional thread-based Apache server software. This is very important for a system such as ours that would ideally be handling thousands of requests per minute as users send time data to and receive predictions from our server.
NGINX communicates to the backend Python code using uWSGI. uWSGI runs our Python code and conducts I/O through a socket that NGINX connects to.

2.3.2 Wait Time Predictor

The wait time projection estimation is created via MATLAB. A Python MATLAB engine is installed on our webserver to run the required scripts and functions to produce the desired results to be passed back to Python function that called the MATLAB function to produce the predicted wait times for the day.

2.3.3 Website UI/UX

The home page for the website hosts a Google Map with a search box, employing the Google Places library and the Google Maps JavaScript API. Respectively, these APIs are used for suggesting establishments to users with an autocomplete feature, and for retrieval and presentation of geographical data about the establishments of interest.

The webpage that presents the proper data to the user uses HTML, CSS, and JavaScript to create a dynamically generated web page that serves the user the correct information in a useful manner. A graph is presented to the user using Charts.js open source JavaScript library to generate the chart in a dynamic modern fashion.

2.4. Algorithms

2.4.1 Wait Time Predictor

There are many ways to evaluate or estimate time series data, which is the type of data we have. The theory and many methods are taken from Peter Brockwell's *Time series: theory and methods*. [11] The kNN Estimator is based on a kNN classification algorithm that returns the k nearest neighbors in ranked order, but instead of returning a mode of the top k, we create our estimation based on the top k nearest neighbors based on our distance metric. Details will be expanded upon in the implementation and evaluation section. The state of the art is shown in [4], which we surpass using the same evaluation metric.

2.5. Evaluation

In order to evaluate the strength and/or accuracy of any prediction produced by any algorithm, we need a measure. We adopt the measure as define in [4], where we take the mean absolute error (MAE) on some given scale, whether it is the next 10 minutes, the next hour, or till the end of the day.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i|$$

Equation 1 - MAE

3. System Architecture

3.1. Data Source Feed

The data source, no matter which it is, will deliver to the database timestamped durations of amount of time spent at a location. It is a one-way communication, since there is no information needed from any other piece of the system.

3.2. Database

We have stored a large number of simulated data points that our prediction algorithm uses. In a working version of the product, we would likely have two databases running simultaneously.

One of the databases would handle real-time storage of user data. Ideally, hundreds or thousands of users would be contacting the service at any point in time, so we would be using a database suited for high performance, in-memory data storage, such as Redis. This would allow collection and handling of large amounts of user data, and would be very scaleable.

The other database would be a standard SQL-based data store. Its purpose would be to hold the data after it has been processed by our system to be in the format the prediction algorithm requires.

3.3. Webserver

NGINX uses an asynchronous, event-driven architecture, that would easily handle thousands of requests per minute as users send time data to and receive predictions from our server. NGINX communicates to the backend Python code using uWSGI. uWSGI runs our Python code and conducts I/O through a socket that NGINX connects to.

We have an API that our data displaying webpage calls on load, with a business associated with it. Which in turn will call our Python code to grab the appropriate data to send to our wait time predictor in our MATLAB engine running in our python environment.

Once the data is received from the MATLAB wait time predictor, the projected data is sent to the webpage for display for the user.

3.4. Wait Time Predictor

When the wait time predictor, which is running within a MATLAB engine, is called from python on the webserver, we are given both the current day's data and the historic data from that location. We then pass this to our internal estimator.

3.4.1 kNNE: k Nearest Neighbor Estimation

Given the current day's data and historic data for a particular location. Our estimator will produce projected wait times for the rest of the day. With a granularity for the time of day of 10 minutes.

Once the kNNE has given us internally our projection for the day, we pass this back to the python function to do with it as is required.

3.5. Website and User View

3.5.1 Search

The home page with which WaitLess users interact contains a Google Map and a search box. Users can query this search box for establishments of interest via title or category, and they may append an address to bias the search results in favor of the address they enter. If no such address is appended to their query, the results will instead be biased in favor of the relevant locations that are on display in the map's viewport at the time of the query. In addition, the search box has an autocomplete feature, which also suggests relevant results inside or near the position of the Google Map's current viewport.

Once the user enters a query, and if there are matching results, photographic markers for the matching establishments will be placed on their locations on the map. The user can then click on those markers to retrieve the WaitLess data associated with them.

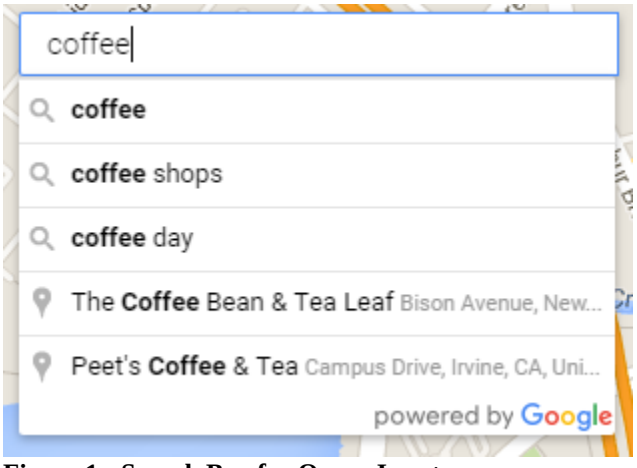

Figure 1 - Search Box for Query Input

A demonstration of Google's autocomplete feature on our map. The map has been cropped out for the legibility of the text shown above, but Irvine was in the viewport of the map at the time of typing into the search box. Here, users are encouraged to choose among categorical searches like "coffee shops" and specific establishments in viewport of the map like Peet's Coffee and Tea.

3.5.2 Data Retrieval for Presentation

When a query is entered and a map icon is clicked, the corresponding page for that result is generated through a GET request; this means that the URL for the page indicates both the establishment name and the establishment ID so that the server can retrieve the related data. In order to generate the right data, placeholders on the HTML page that display the data are filled as follows:

The title placeholder, which states the name of the establishment, is pulled from the Google API and is sent to the GET request via the map interface. It is then placed into the header as shown in Figure 2. Beneath the title header is the relevant wait time data for the establishment.

All of the data related to wait times is provided by the server via the GET request's establishment ID. The server sends the page an array list that contains a wait time for every 10-minute increment of the 24-hour day, all tied to the establishment ID. The array list also contains the current wait time and the wait time 10 minutes from the time of the request. These points are used to fill the placeholders for "Current Wait Time" and "Wait Time in 10 Minutes".

3.5.3 Presentation of Retrieved Data

The WaitLess data presented after a user clicks on a location on the Google Map comprises an estimation of the current wait time, an estimation of the wait time ten minutes after the user selects the location, and a chart of predictions of wait times for the rest of the current day.

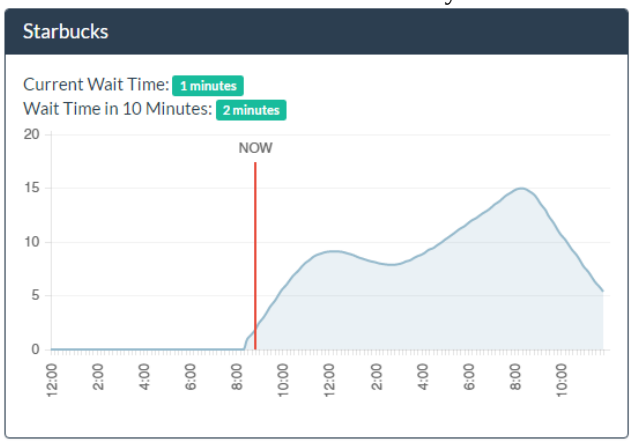

Figure 2 - Sample Prediction Chart, for Starbucks in the UC Irvine Student Center, Queried at 9:00 am

4. Implementation and Evaluation

4.1. Data Source Feed

Ideally the implementation of the data source feed would have been geo fencing using Wi-Fi connection data, however in our quest to get this data from OIT at UC Irvine, we were unable to get the data source in time. As OIT noted to me, the person who would be able to scrub the data so that privacy standards were met, was on vacation until the start of the next quarter.
We needed to still have a data source, so we went with simulated data, where the simulated historic data was created where each day would have a random number of peaks, in the range of 2 and 8 peaks. Where each one would have a mean time randomly between 11am and 10pm, with a standard deviation between half an hour and 5 hours, with a maximum waiting time associated with that peak between 2 minutes and 10 minutes. We then used this data as if it was gathered live from a proper data source.
Every piece of data attached to it would have a business id associated with it, 27 alphanumeric characters as a string, a unique identifier UUID, which is 32 alphanumeric characters as a string, with a corresponding timestamp, and duration.

4.2. Database

Since we are currently only using simulated data, the database implementation is slightly modified for usage of only simulated data for now. We have a standard SQL-based data store. Our table has 5 columns, Business ID (32 chars), UUID (27 chars), a timestamp (timestamp), duration (int).

4.3. Webserver

A major difference between our current version and the ideal production version of the backend software is our algorithms running in the MATLAB Engine. Since MATLAB was more ideal to work in when creating the algorithms, we left that code alone and instead decided to communicate with the MATLAB Engine via Python. The ideal production version would have this MATLAB code translated to Python (or even C++) to gain higher performance and remove the need to communicate between two different runtime engines.

4.4. Wait Time Predictor

The main part of the wait time predictor is the kNNE, which we will now go into further detail. Work is based upon [11] [12] [13] [14] [15]

4.4.1 kNNE: k Nearest Neighbor Estimation

The point of this method is to predict the rest of the day, based upon the entire day up until the current time, and all previous historic data for this location. To this end we are going to identify the k nearest neighbors (kNN) for the current query. [11] [12] Similarity will be defined in a manner most useful for projection estimation. [13] The similarity or neighborness function will be defined as follows,

$$dist(v,w,t) = \sqrt{\sum_{i=0}^{t}([v(i) - w(i)] * g(i))^2}$$

Equation 2 – Similarity Distance Function

$$g(x) = x^3$$

Equation 3 – Weighting Function for Distance

Equation 3 lets more recent data points of the day have a greater weight within the distance function. [14] A simple reasoning would be, the wait times as 3am have little effect on the wait times at 1pm, rather what occurred at 12:50pm will have a greater effect on 1pm than 12:30pm would have on 1pm. The distance function takes the weighted Euclidean distance from the beginning of the day until the current time for a particular historic day, where the weighting is proportional to the cube of the time since the start of the day. [15]

We take the top k, in our system k=5, to produce our projection for the rest of the day.
Once we have our top k, in ranked order based upon the similarity function, the least distance one first followed by the others in ascending distance order. We produce our prediction by taking a weighted average of the top k nearest historic days.

$$projection(d1 \dots dk) = \frac{\sum_{i=1}^{k} di * g(i,k)}{\sum_{i=1}^{k} g(i,k)}$$

Equation 4 – Projection Based Upon Top K Days

$$g(i,k) = [k - i + 1]^2$$

Equation 5 – Weighting Function for Projection

By taking the weighted average, where the weights are proportional to the square of the ranked order of the k nearest data vectors, we are able to produce a projection for the entire day, whose performance will be evaluated in the coming sections. [14] [15]

Note that the current day's data is already known up until the current time. Therefore, the entire day's projection before the current time of day is replaced by the actual wait

time for the current day, but the projection of the rest of the day is produced with satisfying results that we are about to share.

But before that here is an example day where it is currently 11am and we predict the rest of the day, and we see how close we are to being on target.

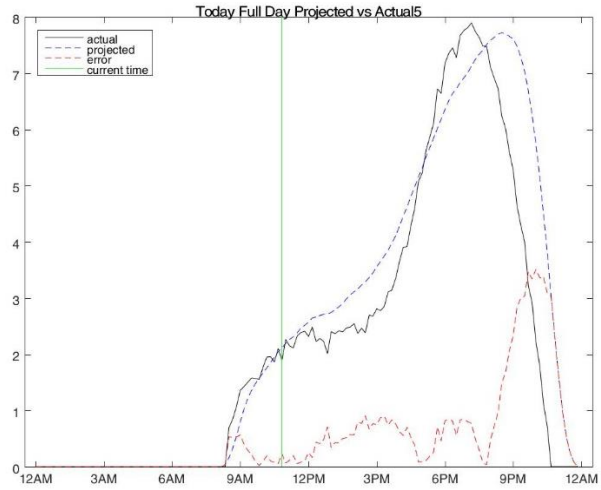

Figure 3 - Full Day Projection From 11am vs Actual Day

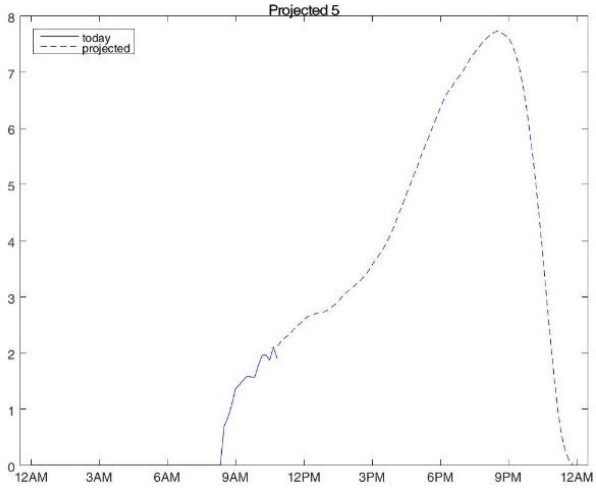

Figure 4 - Projection for Test Day from 11am

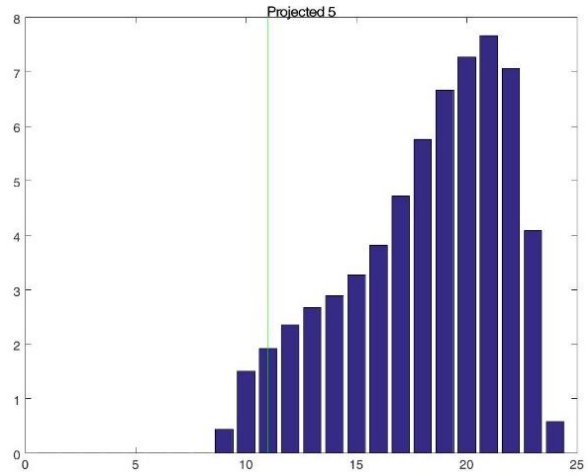

Figure 5 - Projection for Test Day from 11am Bar

4.4.1.1 Timing

Running 160,000 test runs, we are able to create a projection for the entire day, within 3 milliseconds. This is definitely in the realm of real-time and shows the power of streamlined algorithms and their implementations.

4.4.1.2 Performance

Our performance has several metrics, we will be basing all of our performance around the mean absolute error (MAE) of our prediction, where the equation for the is given in equation 1. Our performance markers are the MAE for the next 10 minutes, for the next hour, and from the current time until the end of the day.

A quick note, there is no error for the current wait time, since the current wait time would simply be the latest piece of data we would have from our data source.
Our MAE on average for a 10-minute window is under 1 minute, and our maximal average MAE is 45 seconds.

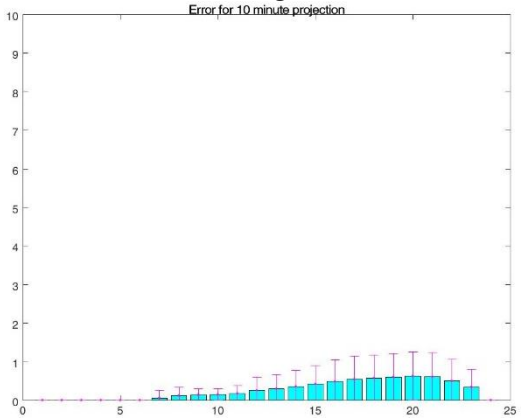

Figure 6 - MAE for a 10-minute projection from different times of day

Our MAE on average for a one-hour projection does not exceed 1 minute and 20 seconds.

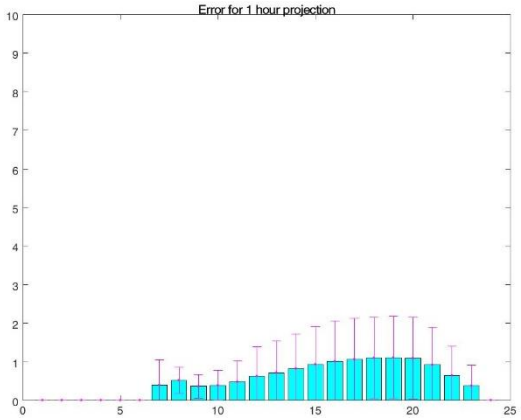

Figure 7 - MAE for a 1-hour projection from different times of day

Our MAE on average for a full-day projection does not exceed 6 minutes, but does improve as the day progresses, which is at it should, as more data is added to the system.

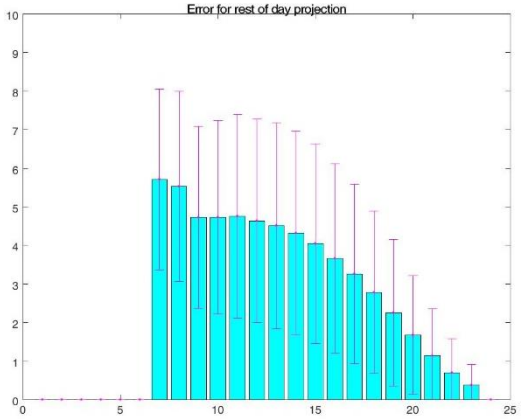

Figure 8 - MAE for a full-day projection from different times of day

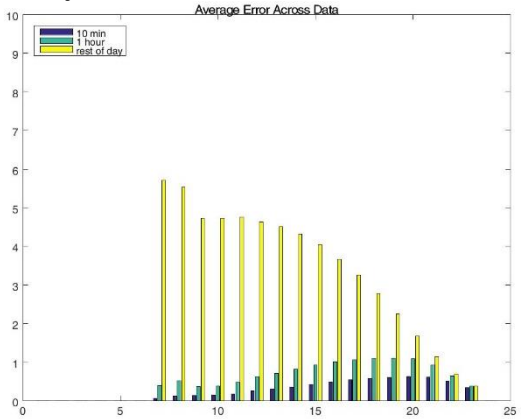

Figure 9 - Grouped MAE from different times of day

4.5. Website and User View

4.5.1 Webpage Map

Owing to the assistance of the Google Places API, our map can reliably retrieve images and GPS coordinates for hundreds of millions of places. [16] But because we do not have data for every dining location known to that library, and because we are using simulated data for the project, we are currently unable to give users wait time information based on real-world data. Nevertheless, our website can still be used as a template for querying and presenting the data we only need to get our hands on it.

4.5.2 Webpage Data Presentation

Figure 2 is a snapshot of the wait times presentation. All of the data related to wait times is provided by the server via the GET request's establishment ID. The server sends the page an array list that contains a wait time for every 10-minute increment of the 24-hour day, all tied to the establishment ID. The array list also contains the current wait time and the wait time 10 minutes from the time of the request. These points are used to fill the placeholders for "Current Wait Time" and "Wait Time in 10 Minutes". These color coded time indicators change colors depending on the wait time; green if the time is less than 10 minutes, yellow if the time is between 10 and 20 minutes, and red if the time is greater than 20 minutes. This is done through a simple inline JavaScript script.

Beneath the wait time numbers is a graph that displays the wait times as a function of time. The aforementioned array list of wait times in 10 minute intervals fills a placeholder on the HTML page. The placeholder uses the Charts.js open source JavaScript library to generate the graphic dynamically.

Finally, there is a red indicator of the current time, notated as "NOW" on the Charts.js graphic, which is pulled from the server's time and adjusted if needed to the users' local time using the browser time.

5. Summary and Further Work

5.1. Accomplishments

We were able to beat out the state of the art [4] when using the same 10-minute window MAE metric. We are able to do it quickly, and without needing access to any person or their smart device directly. We were able to do this without any proprietary technology, rather our own research and bringing together the state of the art and improving upon it.

5.2. Improvements

We were not able to have access to the Wi-Fi data as ideally as we wished, some things were pushed back a bit by OIT,

such as scrubbing of data for privacy concerns. We could in the future work of this research, use the Wi-Fi data to further optimize and enhance our work.

5.3. Future plans

We plan in the future to pursue enhancing this research, with the possible release as either a paper in a reputable journal, or perhaps venture into a marketable product. This will be contingent upon following through with of the outlined improvements stated in the previous section.

6. References

[1] "Average Restaurant Wait Times." FSR Magazine. Web. 31 Jan. 2016. <https://www.fsrmagazine.com/new-restaurant-concepts/study-released-average-restaurant-wait-times>.

[2] "What's The Wait?" What's The Wait. Web. 31 Jan. 2016. <http://whatsthewait.mobi/>.

[3] "Disneyland" Disneyland Wait Times. Web. 31 Jan. 2016. <https://disneyland.disney.go.com/guest-services/download-disneyland-mobile-app/>.

[4] Bulut, Muhammed Fatih, et al. "Lineking: Crowdsourced line wait-time estimation using smartphones." Mobile Computing, Applications, and Services. Springer Berlin Heidelberg, 2012. 205-224.

[5] Davis, Mark M. "How long should a customer wait for service?." Decision Sciences 22.2 (1991): 421-434.

[6] Chen, Junliang, et al. "Tradeoffs between profit and customer satisfaction for service provisioning in the cloud." Proceedings of the 20th international symposium on High performance distributed computing. ACM, 2011.

[7] Bahl, Paramvir, and Venkata N. Padmanabhan. "RADAR: An in-building RF-based user location and tracking system." INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 2. Ieee, 2000.

[8] Haritaoglu, Ismail, David Harwood, and Larry S. Davis. "W4S: A real-time system for detecting and tracking people in 2 1/2D." Computer Vision—ECCV'98. Springer Berlin Heidelberg, 1998. 877-892.

[9] Chadil, Noppadol, Apirak Russameesawang, and Phongsak Keeratiwintakorn. "Real-time tracking management system using GPS, GPRS and Google earth." Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on. Vol. 1. IEEE, 2008.

[10] Namiot, Dmitry, and Manfred Sneps-Sneppe. "Geofence and network proximity." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. 117-127.

[11] Peter J Brockwell and Richard A Davis. Time series: theory and methods. Springer Verlag New York, Inc., New York, NY, USA, 1986.

[12] B. V. Dasarathy. Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[13] Christos Faloutsos. Mining time series data. In SBBD, pages 4–5, 2005

[14] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting, 20(1):5–10, 2004.

[15] Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. Distance measures for effective clustering of arima time-series. In Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, pages 273–280, Washington, DC, USA, 2001. IEEE Computer Society

[16] "Google Places API." Web. 14 Mar. 2016. <https://developers.google.com/places/>.