Predicting Student Demand

Edderic Ugaddan
Lingo Live

October 2016

Contents

1 Definition

1.1 Project Overview

Lingo Live provides customized communication lessons for tech professionals. We specialize in providing engineers in multinational tech companies, many of whom are non-native speakers, a language/communication teacher who knows exactly what they need, anytime and anywhere. Breaking down language and communication barriers helps them become more effective communicators, and therefore, improves their career potential.

It is important for us to make sure that our supply of teachers is able to meet student demand. As part of the tech team at Lingo Live, one of my responsibilities is to provide accurate forecasts on our capacity to handle future students – do we have enough capacity to handle an influx of new students? If so, how much more can we handle? If not, how many more teachers do we need to hire, and what times should they be teaching? Underestimating and overestimating capacity has serious consequences; hiring not enough teachers means that some or many students won't be able to get lessons at times that they want. On the other hand, if we hire too many teachers, many of them might not get enough lessons, and might have to look at other sources of income to supplement the income they get from Lingo Live. Thus, getting accurate forecasts of teacher capacity would help our students and teachers stay happy.

The process of analyzing our capacity involves two main stages. First, predicting student demand given indirect information, such as timezone of these students, which company they are working for, a ballpark number of students that are coming in, and the supposed lesson frequency that people will be taking. We would generate a bunch of possible schedules that these new students might have, based on current student data. Finally, once we have these predicted schedules, we then compare them with current teacher availability to see how much we could handle. My focus for this project is only the first stage – given timezone and company data, could we generate schedules that are representative of the new students?

Data used in this project is from Lingo Live's production database, anonymized to protect students' and companies' privacy.

1.2 Problem Statement

1.2.1 Main Idea

The end goal is to be able to create a model that generates potential students' schedules very well, so that Lingo Live could find the balance between under-hiring and over-hiring teachers. The challenge I would like to address is creating a model that predicts student demand (i.e. when lesson requests patterns are generally likely to occur in the context of the week) for new students – see Table 1 for an example of a student schedule.

The strategy to build this model is as follows. First, I would preprocess the data so that we could represent the lesson requests as sets of "schedules". A

Table 1: Example of a Student Weekly Schedule

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
0:00		✓					✓
0:15			✓				
0:30							
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
23:30							
23:45							

"schedule" in our context would represent the general behavior of when a student takes lessons over a period of weeks. Once that is done, I would build a model and build the benchmarks to test it, which would help us answer the following questions: given access to 'previous' months, could these models predict the 'current' month? Which features make sense to keep? Which ones should be discarded? Which model does the best – and by how much? Finally, I would perform sensitivity analysis to figure out how resilient the model is, so we can assess the validity of the model further, and help us figure out the weakspots of the model that we could improve on in the future.

To simplify the problem of trying to figure out whether one model is better than another, we would bin the weekly schedule into smaller buckets: 0-4, 4-8, 8-12, 12-16, 16-20, 20-0 local time. We don't really care much about whether or not we predict when someone exactly takes lessons – we only care about when someone generally takes lessons. Binning the prediction and actual schedules into 6 buckets helps us figure out when someone *generally* takes lessons.

1.3 Metrics

The error metric e_b for bucket b that we are trying to minimize is as follows:

$$e_b = |y_b - \hat{y}_b(a)| \tag{1}$$

where y is actual and $\hat{y}$ is predicted, and a is a model. More specifically, y_b is the actual number of lessons that belong to bucket b, $\hat{y}_b$ is the predicted number of lessons that belong to bucket b.

The total error, considering each bucket, then is just the sum of the errors for each bucket.

$$TotalError = \sum_{b=0}^{z} e_b \tag{2}$$

$$= \sum_{b=0}^{z} |y_b - \hat{y}_b(a)| \tag{3}$$

where z is the total number of buckets.

We would like to be able to compare how much a prediction in one month compares against other months. The number of lessons from new students per month might vary considerably, as a function of the number of new students signing up, and the lesson frequency. Thus, for a certain month m, and model a, we could average out the total error by dividing by the number of bins x, and get the Mean Absolute Error (MAE):

$$MAE(m, a) = \frac{1}{x} \sum_{b=0}^{z} |y_b - \hat{y}_b(a)| \tag{4}$$

$$\tag{5}$$

We would then measure how the models perform over time. We would split our data into months, and at any point in time, we would use the previous months to predict the "next" month (i.e. time series cross validation)[2]. We would pick the model g that best minimizes MAE on unseen test data over o months:

$$\begin{aligned} g &= \underset{a}{\operatorname{argmin}} \frac{\sum_{m=0}^{o} MAE(m, a)}{o} & (6) \\ &= \underset{a}{\operatorname{argmin}} \frac{\sum_{m=0}^{o} \frac{1}{x} \sum_{b=0}^{z} |y_b - \hat{y}_b(a)|}{o} & (7) \\ &= \underset{a}{\operatorname{argmin}} \frac{\sum_{m=0}^{o} \sum_{b=0}^{z} |y_b - \hat{y}_b(a)|}{xo} & (8) \\ & & (9) \end{aligned}$$

MAE was chosen over other metrics (such as Mean Squared Error, MSE) because it is very intuitive – the metric is in the same units as what we are trying to predict (i.e. number of lessons), so is definitely helpful in terms of interpretability.

2 Analysis

2.1 Data Exploration

I extracted the original lesson request data from the Lingo Live production database by joining several tables, resulting in rows of lesson requests. A lesson request has an id (*lr_id*), and it has a start time (*lr_start_datetime*). The lesson requests start times have a range from March 4, 2014 to November 29, 2016. ($\approx$ 2 years, 8 months). It is associated to the student (via *user_id*), and the student has a timezone (*user_tz*). The student is also associated with a company (via *company_name*). Some lesson requests also have a *cr_id*, which stands for

Table 2: Lesson Request Data Statistics of All Students: Part I

stats	**user_id**	**lr_start_datetime**	**lr_id**
count	81003	81003	80973
unique	1744	26052	80969
top	360	2016-08-24 16:00:00	35944
freq	439	28	2

Table 3: Lesson Request Data Statistics of All Students: Part II

stats	**user_tz**	**company_name**	**cr_id**
count	80388	81003	34821
unique	49	1	34821
top	Pacific Time (US & Canada)	all_students	38346
freq	20128	81003	1

"canceled request id," and exists when a teacher, student, or someone else has canceled the lesson request. See Table 2 and 3 for summary statistics.

user_tz makes use of Ruby on Rails' *ActiveSupport::TimeZone* gem[1]. (See the "Mapping" section online for more information about the possible values). In Table 6, we show the number of unique users per timezone. The top 5 *user_tz* values are Pacific Time (US & Canada), Brasilia, Eastern Time (US & Canada), New Delhi, and Tokyo. This validates my intuition, as most customers are taking lessons in those timezones.

One issue is that there are 940 lesson requests that are missing user timezones; they seem to be concentrated in the year 2014 – not really sure why this is (Table 5). These 940 lesson requests are associated with 88 users. These are discarded from the dataset used by the models since we cannot localize these times.

2.2 Exploratory Visualization

We see densities of lesson requests in the span of the week as a function of timezone (Table 7). We learn a couple of things from this visualization: 1) In general, most lessons start between 6 AM to 10 PM on the weekdays, and 2) timezone might be a good predictor of when people will take lessons. For

Table 4: Sample Lesson Request Data of All Students

user_id	**lr_start_datetime**	**lr_id**	**cr_id**
2077	2015-11-12 10:00:00-05:00	43290.0	33.0
1714	2015-10-06 17:00:00+09:00	35837.0	17985.0
3340	2016-09-30 08:00:00-07:00	118395.0	NaN
3289	2016-09-01 21:00:00-07:00	117997.0	NaN
1279	2015-12-15 14:00:00-02:00	44700.0	26217.0

Table 5: Sample of Lesson Requests that are missing *user_tz* Values

user_id	**lr_start_datetime**	**lr_id**	**user_tz**	**cr_id**
362	2014-10-16 01:00:00.705356	24865.0	NaN	NaN
339	2014-06-19 21:00:00.273685	24414.0	NaN	NaN
361	2014-08-22 00:00:00.978073	27522.0	NaN	NaN
346	2014-08-06 16:00:00.927964	28159.0	NaN	NaN
78	2014-03-13 01:00:00.090054	24440.0	NaN	NaN
340	2014-06-21 19:00:00.822833	27594.0	NaN	NaN
345	2014-05-19 15:00:00.451452	26497.0	NaN	NaN
912	2014-06-26 01:00:00.534694	25951.0	NaN	NaN

Table 6: 19 Most Frequent User Timezones

user_tz	**frequency**
Pacific Time (US & Canada)	431
Brasilia	290
Eastern Time (US & Canada)	145
New Delhi	134
Tokyo	109
London	71
Buenos Aires	65
Chennai	64
Central Time (US & Canada)	46
Jerusalem	36
Mexico City	34
Paris	24
Bogota	20
Seoul	19
Madrid	15
Dublin	15
Mountain Time (US & Canada)	15
Kolkata	11
Guadalajara	10

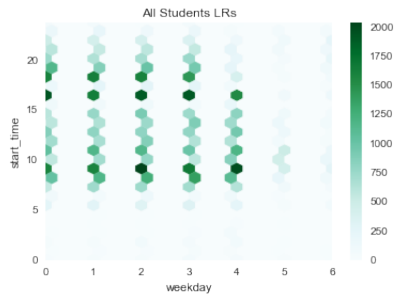
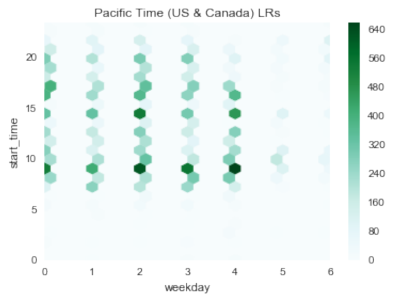
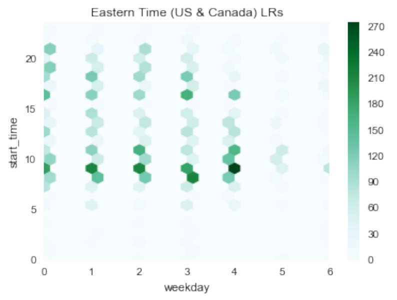
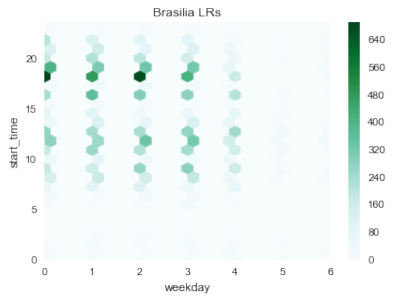
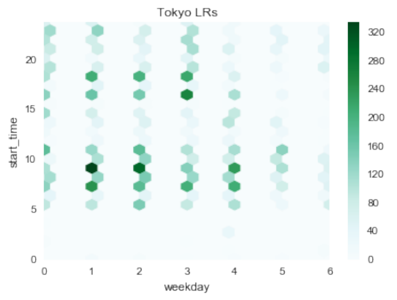
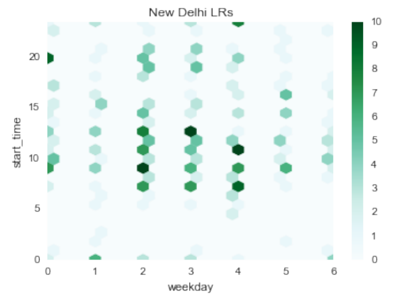
	a	b
1		
2		
3		

Table 7: Weekly Lesson Request Densities by Top 5 Timezone

example, compare Figures and 1b 2b in Table 7 – people taking lessons with Brasilia timezones tend to have most of their lessons happening around 7 PM, but those on Time tend to have most of their lessons at 9 AM and 3 PM. Also, by the same token, Fridays are not as popular with people on Brasilia timezones than people on Pacific Time.

2.3 Algorithms and Techniques

Given that we know how many people will be taking lessons, what their lesson frequency is going to be for the month, and what their timezones are, my algorithm would output a projected number of lessons in each weekly schedule bucket (e.g. 0-4 local time, 4-8 local time, etc.).

The algorithm is quite effective for how simple it is: for all the lessons that ever happened in the training data, put them all in a weekly schedule, and directly compute the probabilities that lessons would be requested for each bin (e.g. 0-4 local time, 4-8 local time, etc.). Then, for each business forecast, we filter the training data to only give us ”relevant” information. For example, if we think that timezone is relevant, then for each business forecast, we bring up data from that timezone, and we calculate the probability that someone will take lessons during each of the schedule bins. We then multiply that probability distribution by the number of students and the lesson frequency. We keep a tally of all the sums.

For instance, let's say we get a business forecast that has 10 students, 2 lessons a week, from Eastern (US & Canada) time, and the training data shows that the probability of a lesson request happening for each bin, is as follows: $[0.1, 0.3, 0.1, 0.2, 0.3, 0.1]$. We multiply that by the number of students and by the number of lessons per week. Therefore, we get $[0.1, 0.3, 0.1, 0.2, 0.3, 0.1] \times 10 \times 2 = [2, 6, 2, 4, 5, 2]$ respectively.

To be more precise, however, we actually smooth out the probabilities by adding a uniform distribution of small values. So from our example, let's say a timezone has a distribution S and we add uniform distribution U:

$$\begin{array}{r} S = [0.10, 0.30, 0.10, 0.20, 0.30, 0.10] \\ + \; U = [0.01, 0.01, 0.01, 0.01, 0.01, 0.01] \\ \hline S + U = [0.11, 0.31, 0.11, 0.21, 0.31, 0.11] \end{array} \tag{10}$$

We normalize it so that the sum is 1:

$$\frac{S+U}{\sum(S+U)} = \frac{[0.11, 0.31, 0.11, 0.21, 0.31, 0.11]}{1.16} \tag{11}$$

$$= [0.095, 0.267, 0.095, 0.181, 0.267, 0.095] \tag{12}$$

Then we multiply the normalized sum by the number of lessons from the forecast. So if there's 10 lessons projected for the timezone of interest, we multiply by 10:

$$\begin{aligned} 10 \times [0.095, 0.267, 0.095, 0.181, 0.267, 0.095] \\ = [0.95, 2.67, 0.95, 1.81, 2.67, 0.95] \end{aligned} \tag{13}$$

Hence we expect about 1 lesson to happen for 0-4 local time, 2-3 lessons during 4-8 local time, etc.

The main point of adding the smoother (i.e. $[0.1, ..., 0.1]$) is to distrust the observed probabilities. For example, just because in our lifetime, we've observed that the sun rises every single day means that the probability that the sun will rise again is 100% [4]. We cannot be fully certain, so we think it's more like 99.99% (close to 100% but not 100%). Similarly, suppose we've observed that for a certain timezone with only a few lessons that no lessons ever happened

Table 8: Dumb Model: Assumes lesson requests (naively) start randomly anytime

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
0:00	✓	✓	✓	✓	✓	✓	✓
0:15	✓	✓	✓	✓	✓	✓	✓
0:30	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
23:30	✓	✓	✓	✓	✓	✓	✓
23:45	✓	✓	✓	✓	✓	✓	✓

from 0-4 AM – it does not mean that the probability that a lesson will happen in the range of 0-4 AM is 0. Smoothing out the observed probabilities would probably smooth out our incomplete observations to match true probabilities.

The advantage of this algorithm is its simplicity, speed, and interpretability. It is very simple and easy to understand, and its' predictions are quite fast, Also, its error is quite small in comparison to the benchmarks, which makes it a good candidate for the problem it is trying to solve.

2.4 Benchmark

For benchmarking purposes, I created two other models that were based on two hypotheses. First, I made a model that assumes that schedules are distributed equally (*DumbModel*, Table 8). Since we know from visualizations that wee hours of the morning are clearly not as likely as the other parts of the day (we also know that weekends are clearly not as popular as weekdays), we expect this model to perform poorly. It is a good to include as a benchmark, because we expect that our more complicated model should be leaps and bounds better than this. Second, I created a model that assumes schedules are equally distributed in the range of hours when most people are active (*SmartModel*, Table 9). We expect that *SmartModel* would be better than the dumb model, but we expect that our model to beat the other two by a significant margin due to the fact scheduling is not randomly distributed by any means.

3 Methodology

3.1 Data Preprocessing

In order to create training and test sets, some data preprocessing had to occur. I transformed raw lesson request data into schedules. I took the first 16 lesson requests of each person and discarded the rest (Table 10). 16 was arbitrary – it could have been smaller. These 16 lesson requests were then projected onto

Table 9: Smart Model: Assumes lesson requests only start when people are generally awake

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
6:00	✓	✓	✓	✓	✓	✓	✓
6:15	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
.	✓	✓	✓	✓	✓	✓	✓
23:30	✓	✓	✓	✓	✓	✓	✓
23:45	✓	✓	✓	✓	✓	✓	✓

Table 10: Example First 16 Lesson Requests of a User

	user_id	**lr_start_datetime**
1	1	2016-09-21 08:00:00-04:00
2	1	2016-09-23 09:00:00-04:00
3	1	2016-09-26 09:00:00-04:00
.	.	.
.	.	.
.	.	.
14	1	2016-09-26 08:30:00-04:00
15	1	2016-09-26 09:00:00-04:00
16	1	2016-09-26 09:00:00-04:00

a weekly schedule. In the weekly schedule (Table 11), I looked for the highest frequency, which I then used to filter out ones that were less than or equal to half of the highest frequency (i.e. ”noise”, see Table 12). Then, I labeled each schedule as being type 1 if there was only one weekly schedule lesson, 2 if there were two entries, etc. We classify the user schedule as belonging to a certain month based on when the first lesson request happened. Thus, User 1 in our training/test data would be represented as data in Tables 13 and 14.

3.2 Implementation

3.2.1 Challenges

One issue was using company name as a predictor variable/feature. I was hoping to examine whether or not knowing about a company could help us more accurately predict when that person will take lessons. Perhaps some companies are more likely to take lessons during the morning than the evening as a function of work culture and positions (e.g. a tech company with mostly tech

Table 11: Weekly Schedule of a User: Intermediary Step

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
8:00	2						
8:15							
8:30	4						
8:45							
9:00			5		5		

Table 12: Weekly Schedule of a User: Final Step

	Mon	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**	**Sun**
8:00							
8:15							
8:30	✓						
8:45							
9:00			✓		✓		

Table 13: Example User Summary: Part I

user_id	**l1_day**	**l1_time**	**l2_day**	**l2_time**	**l3_day**	**l3_time**
1	0	8.5	2	9.0	4	9.0

Table 14: Example User Summary: Part II

start_month	**start_year**	**schedule_type**
9	2016	3

employees might have more flexibility as to when they take lessons than non-tech companies with non-tech employees).

I tried to use it as a variable, however users are not associated with only one company in the system, so we are left with the problem of figuring out which company to assign to the user, which is difficult. Not solving for this problem would mean that in joining the *lesson request* and *company* SQL tables, lesson requests would be duplicated for users that are associated to more than one company, which would skew the results. Their lesson requests would have more representation than others' lesson requests, which might bias the model towards over-predicting lesson requests for those times which have lots of duplication and under-predicting lesson requests for the rest.

For example, consider company A and company B. Let's say company A is "Twitter" and B is "TwitterGlobal". Basing on the naming convention, company A seems like it is going to be a subset of B (Figure 1), but that does not seem to be the case – they both have elements that are not present in the other (Figure 2). What do we do with the people that they have in common? Do we just label them A? Or should they be B? Or do we label them as something else? Due to the time constraints, I decided to tackle this problem in the future as an enhancement.

Another issue was using legacy code as benchmark. There exists code that helps predict student demand. I would have loved to use the code to see how it performs against my model. However, code for it is not maintained and is hard to understand, and the engineer who wrote it is no longer with us. Given time constraints, I chose not to go through with the comparison.

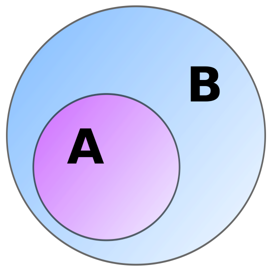


Figure 1: Company A is a subset of Company B

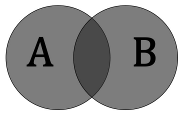


Figure 2: Company A is not a subset of Company B

Table 15: Average Mean Absolute Errors (11 months)

model	average MAE over 12 months
TimezoneProbModel	6.467433
InterpolatedProbModel	6.607820
GeneralProbModel	6.893402
LFProbModel	7.023239
LFTProbModel	7.887060
SmartHeuristicModel	12.062500
DumbModel	23.046296

3.3 Refinement

Several ways were attempted to improve the performance of the algorithm. First, I performed feature selection and created submodels: timezone only (*TimezoneProbModel*), lesson frequency only (*LFProbModel*), lesson frequency and timezone (*LFTProbModel*). It turned out that *user_tz* was the best predictor out of the two variables over several months. Using lesson frequency by itself, and using lesson frequency in combination with timezone resulted in a slightly less performant model. *GeneralProbModel*, which does not take into account either variables when fine-tuning its training data, was also created. It performs slightly worse than *TimezoneProbModel*, but better than the other models. I was quite convinced with using *TimezoneProbModel* as the final model, but thanks to sensitivity analysis (discussed at the Results section), I discovered a flaw. It was mitigated by generating a meta model that makes use of both *TimezoneProbModel* and *GeneralProbModel* predictions. More of this is discussed below.

4 Results

4.1 Model Evaluation and Validation

Over 11 months of test data (2015-09 to 2016-08), *TimezoneProbModel* performed the best (See Figure 3 and Table 15 for details; please note that only the benchmarks and the best performing ones were kept to keep the graph from being cluttered). It is not surprising – looking at Table 7, one could see that timezone does carry information that helps us predict when a student would take lessons. However, after doing sensitivity analysis, I quickly learned that the model itself has a big weakness – predicting schedules of timezones it hasn't seen.

Consider the scenario when a timezone we don't have much experience with (i.e. sample size for the given timezone is very low). For example, at the time of this writing, we only have one user schedule that has "Adelaide" as the timezone. In my sensitivity analysis (*code/sensitivity_analysis.py*), I created fake data to simulate predicting when lesson requests would happen from an un-

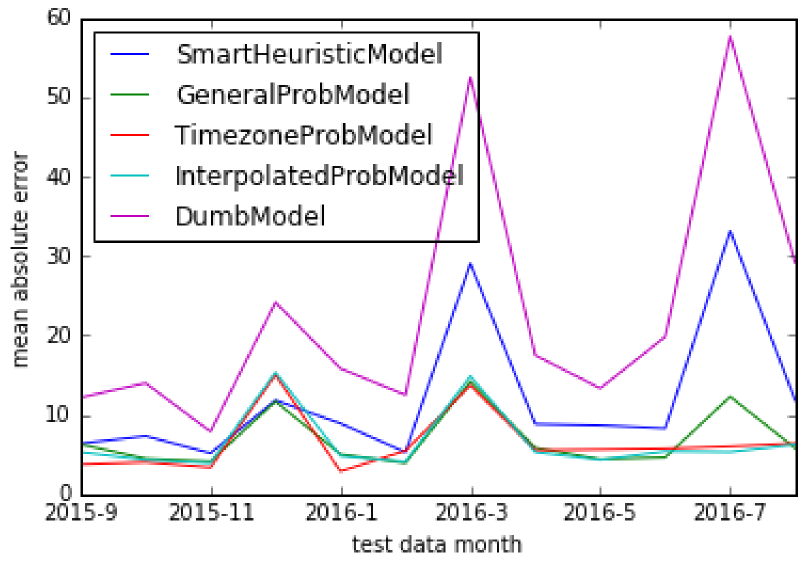


Figure 3: Mean Absolute Errors over 12 months

derrepresented timezone. Specifically, I cloned data from one of the previous months, labeled the clone as belonging to the under-represented timezone (Adelaide). I also then cloned the fake data (labeled as belonging to Adelaide) for the next month after that. Again, the goal is to see how well *TimezoneProbModel* the robustness of the model. Not surprisingly, the errors were huge – by default, *TimezoneProbModel* smooths out observed probabilities, and when observed probabilities are zero (or close to zero), smoothing them out will lead to a random 'flat' distribution, essentially making a guess that even wee hours in the morning are as equally likely to have lessons than other 'normal' times – which we know is not the case.

To remedy this problem, I decided to create *InterpolatedProbModel*, a new model that interpolates between *TimezoneProbModel* and *GeneralProbModel*. The latter assumes that it does not matter what timezones people are coming from. When it gets a business forecast, it discards timezone and uses information from all students' schedules – not just students' schedules that belong to the timezone of interest. It performs slightly worse than *TimezoneProbModel*, but it is still much better than the two benchmarks, and it is able to handle underrepresented cases pretty well. On fake unseen data I created, interpolation decreased error (compared to *TimezoneProbModel*) by about 89.6% (from 172.88 to 17.95 MAE). After cloning the unseen data again, *InterpolatedProbModel* is able to take advantage of *TimezoneProbModel*'s more targeted approach (17.95 to 0.65 MAE), which shows the robustness and adaptability of the model (Table 16).

The general idea of interpolation is the following. When sample size for a

Table 16: Model Performance on Underrepresented, Unseen Data

model name	**MAE on fake_data_1**	**MAE on fake_data_2**
GeneralProbModel	13.319845	11.728293
InterpolatedProbModel	17.947195	0.645111
TimezoneProbModel	172.883495	0.676617

given timezone is small, weight the prediction of *GeneralProbModel* more heavily. When sample size is big enough, weight the prediction of *TimezoneProbModel* much more than *GeneralProbModel*. If sample size is not big enough, but not too small either, we weight the prediction of the two models more equally.

Given the sample size of the timezone of interest, we pass that into a sigmoid function to determine the weighting for each of the two models:

$$\text{TimezoneProbModel weight} = \frac{1}{1 + e^{-\frac{x-80}{25}}} \tag{14}$$

where x is the sample size of the given timezone of the business forecast[3]. This sigmoid function has the useful property of outputting a weight between 0 and 1 (both inclusive). Figure 4 shows an example sigmoid function that could be used for this problem:

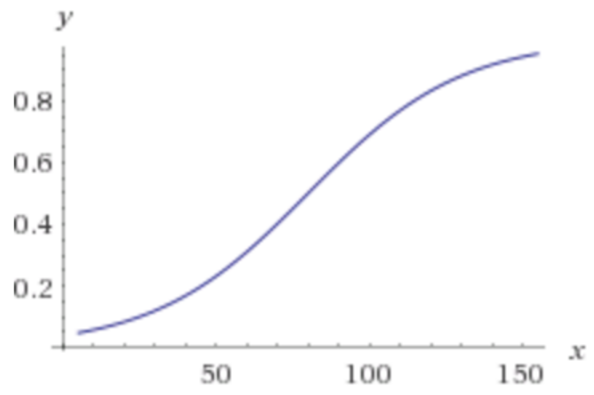


Figure 4: Sigmoid Function adjusted for interpolating between models

So when the lesson request sample size for the timezone of interest is close to zero, *TimezoneProbModel* is weighted weakly. However when the sample size goes toward infinity, the weighting for *TimezoneProbModel* approaches 1. At the halfway point (i.e. where the timezone of interest has 80 lesson requests in the training set), both *TimezoneProbModel* and *GeneralProbModel* are weighted equally. The parameters of the sigmoid function were chosen by running an optimizer. In code, they are labeled *x_offset* and *x_divisor*. *x_offset* directly affects where the cut-off is, and *x_divisor* directly affects the smoothness of the transition between two models. Values for *x_divisor* ranged from 1 to 50, while values for *x_offset* ranged from 100 to 150, depending on what was in the training set.

Timezone carries information that could help predict when lessons will occur. The caveat is when sample size is small – we cannot trust the predictions of *TimezoneProbModel*. However, blending it with *GeneralProbModel* via *InterpolatedProbModel* gives us the best of both worlds – a model that is robust and makes good predictions on under-represented data, yet it is also a model that improves its predictions via the use of more relevant variables once sample size increases.

4.2 Justification

In terms of average MAE, *InterpolatedProbModel* presents a 71.3% improvement over *DumbModel* and a 45.2% improvement over *SmartHeuristicModel*, quite strong results compared to the two benchmarks. As discussed before, interpolation decreased error (compared to *TimezoneProbModel*) by about 89.6% on the example of unseen fake data (Table 16), making it quite robust.

To get an intuitive sense of whether the *InterpolatedProbModel* is significant enough to have solved the problem, we could look at the predictions on test data (Table 17). On this table, the biggest errors (over- and under-estimates) were highlighted, and the biggest of them all is about 30 lessons. Is being 30 lessons off a big problem?

In terms of hours, 30 lessons is equivalent to the following: 30 lessons $\times$ 45 minutes per lesson $\times$ 1 hour per 60 minutes = 22.5 hours. On average, disregarding outliers, a task-based learning (TBL) teacher wants 20.79 hours per week (See Table 18 for teacher desired hours). I think on average, one TBL teacher should be enough. I would conservatively add 1 or 2 more teachers, just in case the lessons requested are supposed to happen simultaneously (as opposed to being spread out).

How likely is this scenario? Given knowledge of how our predictions did on test data, I'd say not very likely. There have been 66 bin predictions made (11 months $\times$ 6 bins per month). Five of those bins were off by at least 15 lessons. Thus, I claim that the probability of being off by at least 15 lessons is about $5/66 = 9.09\%$, which is quite low. I would say that *InterpolatedProbModel* is a significant solution to the problem of predicting student demand.

5 Conclusion

5.1 Free-Form Visualization

The visualization I chose is the error surfaces of the different models: *InterpolatedProbModel*, *TimezoneProbModel*, *GeneralProbModel*, *LFProbModel*, and the two benchmarks *SmartHeuristicModel* and *DumbModel* (See Table 19). Pay

Table 17: InterpolatedProbModel Bin Errors By Month

time	**0-4**	**4-8**	**8-12**	**12-16**	**16-20**	**20-24**	**ss**
2015-9	-2.34	5.85	9.80	-7.31	-6.23	0.23	125.0
2015-10	1.38	2.81	-11.80	1.21	1.33	5.07	120.0
2015-11	-0.18	-2.11	-7.45	6.61	-1.32	4.45	55.0
2015-12	3.88	0.41	**-30.56**	**-17.63**	**33.26**	10.64	199.0
2016-1	-1.23	2.29	-10.74	-1.65	5.44	5.89	137.0
2016-2	2.30	-3.16	8.12	2.12	-3.95	-5.42	141.0
2016-3	4.02	**20.75**	**-27.48**	8.47	12.10	**-17.85**	485.0
2016-4	1.49	5.10	-6.30	-5.05	-4.39	9.16	119.0
2016-5	1.47	5.14	-10.65	0.89	5.89	-2.73	120.0
2016-6	1.25	4.08	11.51	-4.27	-10.48	-2.08	173.0
2016-7	-0.86	9.87	-4.15	-1.34	6.79	-10.32	548.0
2016-8	-2.69	4.44	5.90	-10.91	-5.25	8.51	239.0

attention to the contour maps – sections of the surface that are close to zero are marked gray; ones that overestimate are orange/red, but ones that underestimate are light blue/dark blue. A good model in general would have less blue and orange areas.

Out of all the options shown, *InterpolatedProbModel* seems to have the least amount of blue/orange areas, so it's predictions in general are more spot on than other models. *GeneralProbModel*, for example, has a bunch more orange spots, which means that it's overestimating more on average a bunch of sections than *InterpolatedProbModel*.

We could also see that the chosen model, *InterpolatedProbModel*, is much better (smoother) than the benchmarks*SmartHeuristicModel* and *DumbModel*, judging by the jagged shapes – these stalactites are huge signs of underestimation.

Another visualization that provides evidence of *InterpolatedProbModel* being a good model is the error bars (Table 20). One can see that the errors for each bin, on average, are quite close to zero, specially when compared to the two benchmarks (*SmartHeuristicModel* and *DumbModel*). On average, the two benchmarks tend to severely over- and under-estimate unlike *InterpolatedProbModel*. The errors also have a much smaller standard deviation than the benchmarks.

5.2 Reflection

In this project, I was able to create a model that predicts when students will be taking lessons pretty well, given that we have information on their actual lesson frequency and the timezones that they came from. The model is essentially an ensemble of two submodels: *GeneralProbModel* and *TimezoneProbModel*. When asked to predict when a student will be taking lessons, the first makes a prediction without paying attention to the what the timezone is. The training

Table 18: Task-Based Learning Teacher Desired Hours

desired hours	**desired hours**	**desired hours**
-30	16	30
0	17	30
5	18	32
6	20	35
6	20	35
6	20	40
8	20	40
10	20	40
10	20	45
10	20	50
10	20	NaN
10	20	NaN
10	23	NaN
10	24	NaN
11	25	NaN
12	25	NaN
12	25	NaN
15	25	NaN
15	25	NaN
15	25	NaN
15	25	NaN
15	28	NaN
15	30	NaN
15	30	NaN
15	30	NaN
15	30	NaN
15	30	
15	30	
16	30	
16	30	

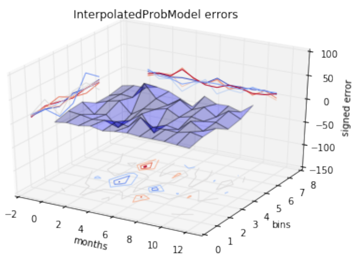
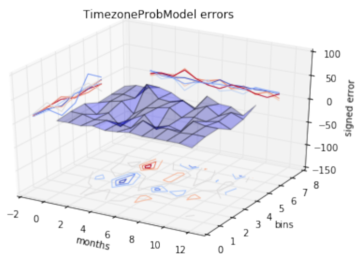
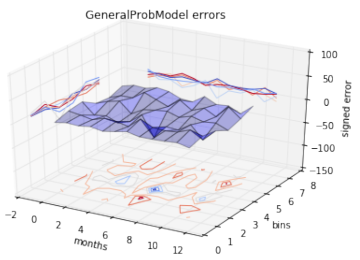
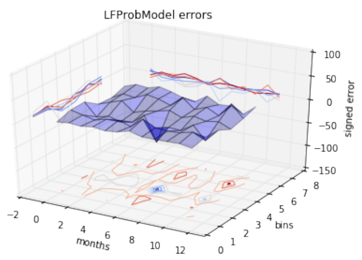
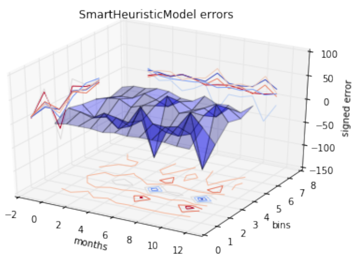
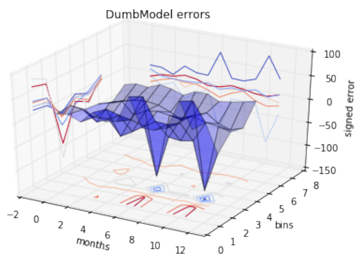
	a	b
1		
2		
3		

Table 19: Error Surfaces of Different Models

set consists of schedules without any care about timezone, which has the benefit of being robust in predicting schedules of a "new" timezone. On the other hand, the latter does make extensive use of timezone when predicting when students will be taking lessons and generally gets better results than the former – it assumes that people in one timezone might not be having the same scheduling patterns as others. The problem with this approach, however, is when the model encounters unseen data (i.e. from a "new" timezone). When the sample size is really low (e.g. 0), it's guess is basically random, which we know performs poorly. Thus, in this project, the final model is one that interpolates between two models depending on sample sizes of the different timezones based on the business forecast. This gives us the best of both worlds.

One interesting thing I learned about this project is the Law of Diminishing Returns. In the fitting step of *InterpolatedProbModel*, I wanted to figure out if there's an optimal cut-off point between the two models, and how "soft" or "hard" the cut-off would be. To do so, the training data was subdivided into two groups: sub-training and validation. Sub-training data was used to

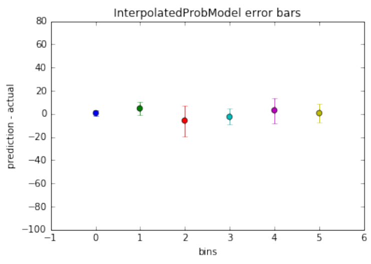
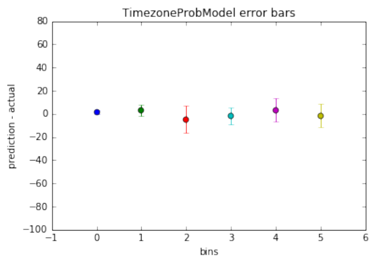
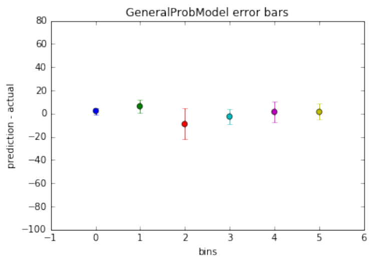
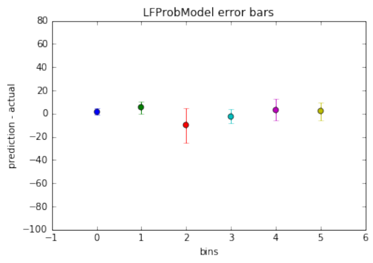
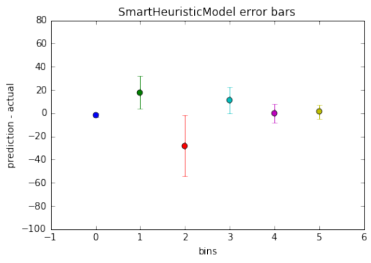
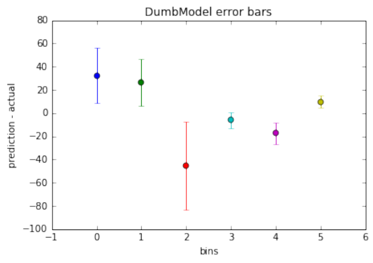
	a	b
1		
2		
3		

Table 20: Error Bars of Different Models

predict error on the validation set, which is the three most-recent months in the training set. Then, I used an optimizer for a number of iterations to find the best cut-off, which took several hours. What I optimized for is the mean absolute error multiplied by the standard deviation of the signed error of the bins. It improved the error-surface by a little bit (Table 19). Again, the error surface has more gray areas, indicating that in a lot of predictions, the error is close to zero. What I found was that the cut-off ranged between about 100 and 150, and that it also varied from "quite soft" to "hard", depending on what was in the validation set. However, it might not have been worth it, considering the amount of time waiting for a result. *InterpolatedProbModel* without optimizing for the "cut-off" and "hardness" of the cut-off had about the same performance in terms of MAE.

Overall, I am quite happy with the performance of *InterpolatedProbModel*. It will be used as part of a whole end-to-end system that would be used to eventually predict when we need to hire teachers.

5.3 Improvement

This project, as described before, is supposed to be part of an end-to-end system that would help forecast how many more teachers we need to hire in the future. It would greatly be enhanced once the remaining parts of the system are built, such as having an accurate model of what the business forecast would be like (i.e. when business people say they think 50 students from Pacific Timezone are going to join next month, we should also consider organic growth (and churn) for that timezone and others). That would help create an accurate business forecast, which would then be fed to the model to generate student demand. Similarly, another thing that would improve this project is if the student demand forecast is then connected with the recommender algorithm so we could run simulations on when we think teachers would be needed next.

6 Appendix

6.1 Code Overview

Scheduling Patterns.ipynb is an iPython Notebook that goes opens the anonymized lesson request data (*data/export_lesson_requests_2016_09_08.pkl*) and aggregates them into schedules for each user (*unique_user_summaries.pkl*). The predictions of different models over months of test data is done through *code/main.py*, which uses *unique_user_summaries.pkl* to generate the predictions (*x_model_errors.pkl*). Sensitivity analysis is done through *code/sensitivity_analysis.py*. Graphs of errors and the accompanying tables were then generated through *Predictions.ipynb*, which makes use of *x_model_errors.pkl*.

References

[1] David Heinemeier Hannson. Activesupport::timezone. `http://api.rubyonrails.org/classes/ActiveSupport/TimeZone.html`, 2016. [Online; accessed 12-September-2016].

[2] Rob J Hyndman. Why every statistician should know about cross-validation. `http://robjhyndman.com/hyndsight/crossvalidation/`, 2010. [Online; accessed 11-September-2016].

[3] Wikipedia. Sigmoid function. `https://en.wikipedia.org/wiki/Sigmoid_function`, 2008. [Online; accessed 6-October-2016].

[4] Wikipedia. Additive smoothing. `https://en.wikipedia.org/wiki/Additive_smoothing`, 2014. [Online; accessed 28-September-2016].