Smunch
Internship 2017 (summer)

Wednesday July 12 Noah M. Schumacher

Order Predicting

This document will go over the factors that lead to a user ordering Smunch or not. It will detail possible approaches for using collected data to predict:

1. If a user will order on a certain day.
2. How many orders a company will place on a given day
3. How many orders a certain restaurant can expect on a given day.

It will also explore the timing of orders. By this I mean at what time a company historically placed 90% of its orders. From this historical data we can predict when 90% has been placed. Hopefully this information will provide insights to number of total orders placed and can be incorporated in the prediction modeling.

1 Preliminaries: User Factors

From these factors their are two possible routes for determining expected order. Use of a multiple regression algorithm (most likely linear) will take in these weighted independent variables and produce an expected total order count from a company. Or, a decision tree which would ultimately lead to a yes or no decision for the User ordering or not. I think that using a multiple regression model to predict the total company orders would be the best method. To start off only around 5 factors will be necessary for individual user predictions. I believe the five most important factors, that continuously update are:

- Restaurant (companies previous order totals from restaurant)
- Day of the Week
- Rating (companies previous order totals from restaurants with given rating)
- Weather
- Delivery Time

Note: I mention continuously update because some of the factors might not play a different role everyday but are a one time factor that underlines a trend; e.g. A company location might be in an area with limited food options which leads to a higher number of employees using Smunch and also a higher density

User order factors.

Factor	Strength	Description
Restaurant	*****	The food offered on given day (ratings, type, ...).
Delivery Time	**	When the food is expected to be delivered.
Friends/Coworker Orders	****	If others in office are ordering.
Weather	***	Weather forecast.
Day of the Week	****	What day of the week it is.
Age	**	Age of User.
Company location	****	Company close to good food?
Salary	**	Salary of User.
Hours per day	*	How long User works per day.
Order Trend	*****	Trend of previous User orders.
Rating	***	Users recent and/or overall ratings.
Type of Job	***	Users job category. (sales, dev, ops, ...)

Table 1: User order factors table showing possible parameters/variables and guessed weighting strength

of orders per day. But this factor does not continuously change and therefore is captured in the other relevant information, such has order trend.

Difficulties:
While the idea of a regression model seems easy we run into problems because these factors iterate and update in all different ways. So instead a of a unified regression such as the one below:

$$OrderPercentage(x_1, x_2, x_3, x_4, x_5) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5$$

Where the β parameters are the partial regression coefficients and x's are the predictor variables. It might be more meaningful to gather data independently and then combine based on the given parameters for a day. Perhaps this is best explained in an example.

Say we are given the table below . From the data we can gain insights to the future week by taking various trends from previous data like so.
From each of these linear fits we can get the expected orders for the instance which this example is next Monday. Apply a weighting equation to average all of the individual predictions:

$$PredictedOrders = \frac{Prediction_1 * Weight_1 + Prediction_2 * Weight_2 + Prediction_3 * Weight_3}{Weight_1 + Weight_2 + Weight_3}$$

Observation	DayofWeek	Restaurant	RestRating	CompanyOrders
1	Mon	Rest1	4	20
2	Tue	Rest2	2	12
3	Wed	Rest3	3	48
4	Thu	Rest4	4	10
5	Fri	Rest5	3	23
6	Mon	Rest6	5	25
7	Tue	Rest7	2	9
8	Wed	Rest1	2	40
9	Thu	Rest2	4	26
10	Fri	Rest3	3	18
11	Mon	Rest4	4	49
12	Tue	Rest5	2	14
13	Wed	Rest6	5	38
14	Thu	Rest7	3	25
15	Fri	Rest1	4	29
16	Mon	Rest2	*4*	
17	Tue	Rest3	*3*	
18	Wed	Rest4	*4*	
19	Thu	Rest5	*2*	
20	Fri	Rest6	*5*	

Table 2: Table shows data for 3 work weeks and then set schedule for next week. Italicized numbers are the most current rating of the restaurant. We want to predict the Company Orders for Mon-Fri.

Figure 1: Example graphs from three parameters

For example from the trend lines on each of these graphs we get P_1 = 60.33, P_2 = 54, P_3 = 43.9 for the upcoming Monday. With weights set to 1

$$PredictedOrders = \frac{60.33 * 1 + 54 * 1 + 43.9 * 1}{1 + 1 + 1} = 53$$

However if we decide we find the rating to be more impacting than the other factors we can up the rating and we can assign its weight to say 3 and get:
For example from the trend lines on each of these graphs we get P_1 = 60.33, P_2 = 54, P_3 = 43.9 S, with

weights set to 1

$$PredictedOrders = \frac{60.33 * 1 + 54 * 1 + 43.9 * 3}{1 + 1 + 3} = 49$$

As we can see the weighting allows for dynamic changes to the "strength" of each of the factors. This is also where machine learning will play a role. By continuously checking the predicted result with the actual the algorithm will continuously build precision by adjusting the weights. While this is a simplistic model it is very scalable for more parameters and also very flexible in that it is isolated from each individual factor.

Foresights:

- Need to scale each prediction for company size.
- Does not take into account direct user feedback rating but rather the companies and total customer base into account (has pros and cons)
- Could use lot of computational memory (still need to research this field)
-

2 Initial Trials for Company Orders

Here are the first attempts at predicting the company orders on any given day. For this first trial I simply used the manually entered Smunch excel file and two python scripts. Each script reads in a dataframe of information from the excel file and outputs a prediction. The prediction is based on a linear regression model of previous data. The best fit predicts the number of orders for the upcoming day that fits its criteria. The two factors I have taken into account for these initial tests are Day of the Week (mon, tue, wed, thu, fri) and the restaurant supplying the food. The prediction number is based on a an evenly weighted average of the two factors. I gathered 16 data points mainly using companies that have had more orders and restaurants that have been active for longer to not have overly skewed data. The results are detailed in the above table.

Initial Prediction Tests

Company	Day	Restaurant	Predict Range	Prediction	Actual
Book a Tiger	Fri	Chupenga	[3, 8]	5.5	11
Mahren Immobilien	Wed	Berlin Burrito	[12, 13]	12.5	13
Smava ff	Thu	Fatoush	[6, 7]	6.5	5
Kontist	Wed	Chutnify	[13, 15]	14	14
Dojo Madness	Fri	Sam Yuk Gu	[5, 6]	5.5	9
Smava ks	Tue	W Der Imbiss	[17, 20]	18.5	18
Smava ks	Fri	Chupenga	[13, 21]	17	17
Wayfair	Fri	Chupenga	[55, 58]	56.5	41
Mahren Immobilien	Thu	Grindhouse Burger	[13, 16]	14.5	15
Kontist	Thu	Grindhouse Burger	[12, 15]	13.5	15
kiwi.ks	Thu	Grindhouse Burger	[26, 38]	32	26
Freshdesk	Thu	Chutnify	[17]	17	16
Mediamath	Wed	Chupenga	[12, 13]	12.5	11
Liqid	Fri	N/A	[27]	27	32
Lofelt	Fri	Son Kitchen	[7, 17]	12	12
Glispa	Wed	Chupenga	[9, 16]	12.5	10

Table 3: Depicts initial trials for initial multiple linear regression model algorithm. Companies and Restaurants were chosen because of their frequency in orders

3 Adjustments

Having implemented a basic algorithm that reads in company, day, restaurant and outputs an expected order count, there is a clear need for adjustments. From the initial predictions just based on the above criteria it was evident that some form of metric needed to be taken into account for the recency and frequency of ordering. Not all companies order on all days of each week even though they have the ability to. Some are even more infrequent ordering maybe a differnet number of days a week or month. We as Smunch might not know what day they decide to order very far in advance and therefore need a probabilistic approach to predicting when they will order and if so how much. Breaking down companies into categories with descending predictability

1. Companies that order everyday
2. Companies that order on same day/s
    (a) of the week

(b) of the month

3. Companies that order same number of day/s
   (a) a week
   (b) a month
4. Companies that order different number of day/s
   (a) a week
   (b) a month

From these categories we notice that a need to predict if a company is going to order or not starts at the third category. Category 1 and 2 are very much so "set in stone" with regards to the company ordering or not.

Process: Find companies that do not fit into categories 1, 2. Place these companies into respective categories $[3a, 3b, 4a, 4b]$. For each category use past days of order to predict which day or days will have orders in upcoming month/day. This can be done by simply number of orders on a given day (mon-fri) or (1-31). For example, suppose we have CompanyA with orders on the following days for a month.

Prediciting Day of Week

Week	Mon	Tue	Wed	Thu	Fri
Week 1	X		X	X	
Week 2	X	X		X	
Week 3	X	X	X		
Week 4	X			X	X

Table 4: X represents that the company ordered on this weekday.

Assuming this was the only information from CompanyA we could gather the probability that CompanyA would order on which days in say Week5. Based on this data the

Prediciting Day of Week

Week	Mon	Tue	Wed	Thu	Fri
Week 1	100%	50%	50%	75%	25%

Table 5: The percent chance that CompanyA will order on each day for the upcoming week.

The idea here to present the percent chance that each active company will order on any given day, while also separately presenting the predicted number of orders that company will have if they do order. An admin will then be able to selected or deselect a company at their choosing.

Procedure: In order to make these predictions I will need to sort the companies into their respective categories. The below decision tree details the method of placing each company in their category. Note, that once if a company is placed in one branch they are then excluded from all other branches.

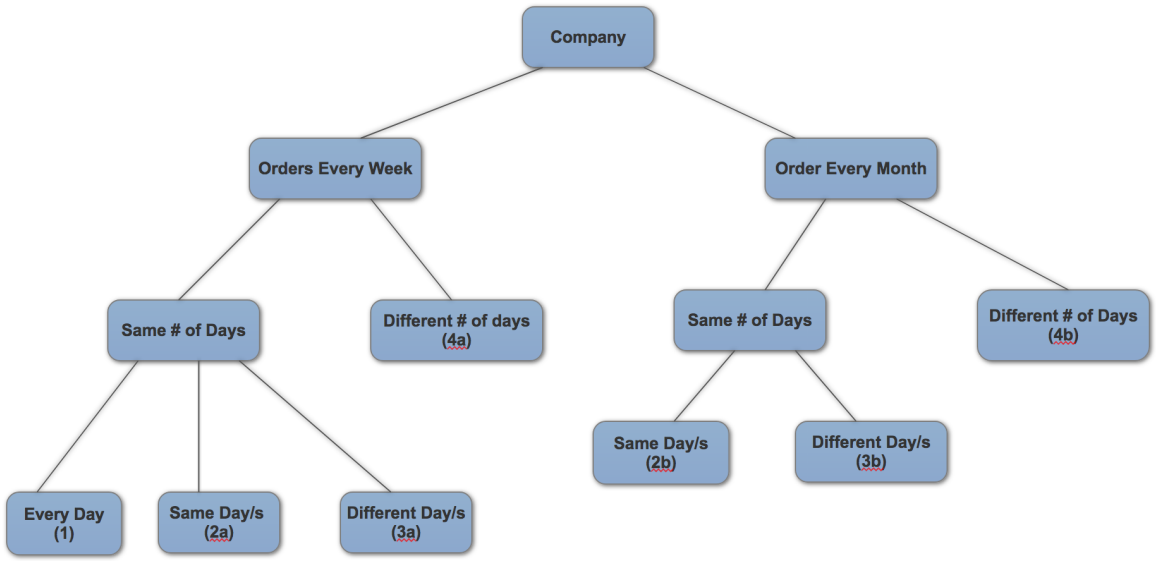


Figure 2: Tree allows for quick categorization of company based on yes or no answers.

In this decision tree it also makes sense to check each branch farthest to the left and then move to the right. For example using the same CompanyA, we first check that they have "Ordered Every Week" which they have (no empty weeks in table). We then check that they have ordered the "Same # of Day/s" which they also have (3 days a week, every week). From here we can store the number of orders and if it is not 5 then we know it is not everyday. Checking "Same Day/s" we see that it is not same days, and by elimination we place CompanyA in category 3a. Doing this for all companies that will efficiently decide what companies need to have an order or not probability. Note, their will be companies that do not fit into "Every Week" or "Every Month", these companies are to infrequent to make any logical prediction on the day they will order and will have to be incorporated on a manual basis.

1. Get first and last order placed.
2. Check for orders every week.
   (a) If orders every week:
   (b) Find if number of days a week is consistent
   (c)
3. Companies that order same number of day/s
   (a) a week
   (b) a month
4. Companies that order different number of day/s

(a) a week

(b) a month

4 Moving to Web Application

Initial thoughts: I am at a crossroads in deciding whether or not it is feasible for me to deploy an active prediction table on the website or not. There are a few subtle difficulties that could make this infeasible for me.

1. Do not know how to query the database for the information I need.
   (a) This could be a quick solve seeing as I have had some luck getting a few parts of the information I need; such as company names and total orders for all companies on a given day.
   (b) I also do have the SQL quarries for companies ordered in the past month and some others that will help.
   (c) Where I run into trouble is both in putting in place the SQL quarries and finding my own information.
   (d) The formatting of SQL quarries is not know to me. List? Array? Series?...
2. Working with the data from the Db Quarries.
   (a) I am confused on how to work with the data that is given to me from the Quarries. Writing all of the logic in an embedded ruby(html) page seems incorrect and very ineffective.
   (b) Ruby has little native libraries for numerical analysis. Python has a lot and I have pre-written code that works but no idea how to implement it into a rails web application with all the libraries needed web application.
3. In the process of building this web application I don't have any test information to see if the values I am acquiring are correct, or at least am very confused on how to check the data.
   (a) There might be a simple fix tot his but if I cant check my work easily and frequently by checking outputs on the expected data its slows down progress.

Updates: I have come to learn that it is possible to simply query the DB straight from a python file using the pycopg2 library. I have written a basic python file that returns all the unique companies from the the trials database on my local machine using and SQL query. Moving forwards I should be able to produce everything that was done before in a more efficient way because of the sql querys. Once I have replicated the same steps using the database I will have to figure out a way to incorporate this into the web application.
Working in Smunch DB: I have just finished a basic python script in /smunchData/dbPredictions/sqlPredictions.py which executes the same prediction as the prediction based on the smunchData.csv file. It is much faster at doing this however and far easier to access information. Now that I have completed the basic prediction process I am debating which direction to go now. I have thoughts that I should form the sqlPrediction.py file into a class structure to make it more object oriented and "malleable". I am also thinking this might be a good time to incorporate some machine learning aspects into the prediction, however I want to have a finished product that Smunch can use and making it into a class might be more useful right now.

End of Day Notes Aug1, 2017:
I have finished the first working version of the company prediction file. This file right now need to be given a specific Day of the Week a specific company and a specific restaurant to calculate the prediction. In future it will read in live data from the site to make all these predictions.
I have also started a new file that will eventually display the avg times when 50%, 75%, 90% of the orders are historically placed for a specific day. Notes on this

1. Need to find where the specific time an order is placed is located in the db
2. Think about what parameters might play a role in these times; [DofW, Weather,...]
3. More specifically need to find the most efficient SQL query that does as much as possible for running through all the orders because it will take a while. (Hopefully it will be a one time run though)
4. Figure out a way to aggregate the data; by company, all company's togeather, all days together
5. Make sure to show std deviation and try and be aware of sqewing information... possibly bulk orders?

Aside from the SQL queries I was thinking it would be good to centralize this work into a python class or classes to help deal with the size. I am not sure if this is necessary but would be great practice. Could be a waste of time though. Looking ahead I need to get a better understanding of how I am going to implement this information on the website and or API. More specifically how I can acquire the live data, such as company A ordering on Wednesday from Restaurant B...

Update Aug 4, 2017:
I have just committed the first finished version of the files that get all the average time a company has placed 75%, 90%, 95% of its orders. This file is exported to a csv named order "placedpercentagetimes.csv" in the dbPredictions folder. I have also committed a file that reads in this csv and plots the standardized distribution of for each percentage. The information gathered in this file can be incorporated into the predictions of a company orders and will work to refine and more accurately predict the orders as the cutoff time approaches. Important to note that the file stored only has the average time that x% orders were placed but the script does at one point find all the order times for x% and this information can be used in the same presentingFile. Might be beneficial to break down each company into DoW for predictions.

Update Aug 7, 2017:
I have completed writing the prediction file and the time for x% of orders to be placed, both for individual companies on specific days and for all companies on all days aggregated. Moving forwards here is what needs to be done, approximately in order

1. Push these files to the server and write live queries for predicting assigned companies and so on.
2. Incorporate x% of orders completed times into prediction algorithm.
3. Make sure to weight each parameter appropriately.
4. Output prediction information to website GUI, (possibly on the side of the delivery dates, menu page).
5. Incorporate some sort of self correcting machine learning for weighting.

Incorporating x% of order time:
Strategy for incorporating the information gained in the x% of total orders completed file. This script takes in a company and a day of the week and creates a bell curve for 75%, 90%, 95% of orders completed. For example Mahren Immobilien looks like this.

Day of Week	% Ordered	Avg Time Before Closure	Std (hours)
Monday	95%	4.14	6.24
	90%	6.38	8.84
	75%	12.67	12.63
Tuesday	95%	7.65	9.01
	90%	10.85	9.18
	75%	20.63	10.56
Wednesday	95%	6.29	7.29
	90%	12.63	10.76
	75%	24.4	13.45
Thursday	95%	7.65	10.16
	90%	16.17	12.45
	75%	24.39	17.02
Friday	95%	11.91	13.2
	90%	15.36	13.57
	75%	22.82	17.15

Where as Smava KS has the table:

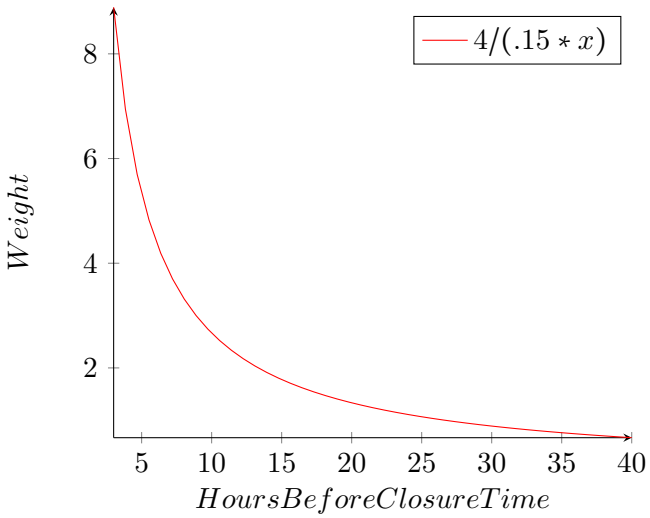
Day of Week	% Ordered	Avg Time Before Closure	Std (hours)
Monday	95%	3.71	2.99
	90%	4.61	3.42
	75%	13.7	10.59
Tuesday	95%	3.63	3.33
	90%	7.02	6.62
	75%	18.15	7.95
Wednesday	95%	4.41	4.59
	90%	7.89	7.09
	75%	19.22	8.7
Thursday	95%	6.31	6.28
	90%	10.3	7.9
	75%	22.2	9.66
Friday	95%	8.2	7.62
	90%	11.39	9.43
	75%	24.81	14.54

Now lets assume the day we are interested in is Tuesday, and Chupenga has been assigned. We know both of these companies are ordering. Without incorporating time we would predict that Mahren Immobilien will order 9, and Smava KS will order 17. Towards the end of the day you see that these predictions might be wrong. If it is 18:00 on Monday then it is 11.5 hours from cut off time. Looking at Mahren Immobilien on Tuesday we see that 10.85 hours before closing they generally have ordered around 90% of their meals. Where as Smava KS is somewhere in between 75% and 90%. Lets take a sharper look at Mahren first

and see how we might incorporate this information into our prediction. Looking at Tuesday 90% we see that the avg time is 10.85 hours before close off but the std is 9.18 hours, not very precise. We want our weighting of this to reflect the closeness to the avg and the precision of the std. A simple weighting formula could be something like so

$$W_T = \frac{5}{w_t(ActualTime - AvgTime) * \frac{1}{w_1} + (Std * \frac{1}{w_2})}$$

Where w_1, w_2 are helping to weight each of the individual factors. I also placed them in the denominator so that increasing their value increasing their weighting strength, avoiding an inverse proportionality. w_t is used to scale the closeness of the time the prediction is made to the closure time. The structure of this weighting makes its stronger when the Actual Time is closer to the Avg time and std is smaller or more precise. An ideal weighting function would look like so:



This graph shows a couple key things. First the weight is almost negligible until we get around the 24 hours mark, at which point it starts to play a meaningful role in the prediction. Second is that as we get close to 0 hours before closure time the weighting blows up. This makes sense because this parameter is the only one taking the current orders into account and the closer towards the cutoff time we get, the more likely it is almost all the orders have been placed. Third and more subtly is that this shows the desired type of function used for weighting is a Reciprocal Function, indicating that the above equation is of sound form.

So lets assume at 18:00 on Monday we only have 5 orders from Mahren. Now looking out our 90% orders time we would assume, if it was perfect that the total orders would be 5/.9 = 5.5. Now remembering our prediction formula from the begging of this document we can do something like this.

$$PredictedOrders = \frac{(13 * W_D) + (6 * W_R) + (5.5 * W_T)}{W_D + W_R + W_T}$$

Assuming we have $W_D = W_R = 1$ and $w_1 = w_2 = 1$, from out previous formula we would have

$$W_T = \frac{1}{.65 + 9.18} = .102$$

This would give us a final prediction of

$$PredictedOrders = \frac{13 * 1 + 5 * 1 + 5 * .102}{1 + 1 + .102} = 8.81$$

This hardly changes the prediction at all. Assuming that the x% information might be quite a strong factor we can raise the weight of it by increasing w_1, w_2. Lets have the new W_P equation be

$$W_T = \frac{1}{(.65) * \frac{1}{4} + 9.18 * \frac{1}{5}} = .5$$

Changing the predicted orders to

$$PredictedOrders = \frac{13 * 1 + 5 * 1 + 5 * .5}{1 + 1 + .5} = 8.1 \approx 8$$

Now our prediction has changed, -1 meal. While a one meal difference might not be a lot doing this for bigger companies could drastically change the prediction and continuously increase accuracy. Also aggregating over all companies ordering will make a signifigant impact on the total prediction for a day. The amount of weighting possibilities makes things complicated but also allows for higher precision by running tests on previous data. The downfall of this form of prediction is the inherent necessity to be close to closure time. Incorporating time in almost all cases wont be useful until we are within 25 hours of this closure time.
Now turning out attention to Smava KS we encouter the second difficulty in incorporating time%. Using the same information as before, Tuesday orders, 18:00 Monday afternoon, predicted orders = 17. 18:00 means we are 11.5 hours from closure time but looking at the chart that amount of time seems to be right in between the 75% and 90% Avg time.

Putting it all together:
Moving into my final week and a half at Smunch the sole goal is to move the prediction algorithm to the live server. What this means in more detail:

1. Enclose prediction algorithm and all dependencies in docker image.
2. Once image works locally move to live server.
3. Implement live query file to get information from server in real time and returns:
   (a) Day of Week.
   (b) Company ID.
   (c) If restaurant assigned to company, Restaurant ID.
   (d) Current orders placed by company.
4. Implement smart caching to only update prediction when change to parameter above has been made.
5. Leave detailed report of how prediction works and what errors might through for Torsten to work with.