Sam Bhagwat's Uber Demand Data

Overview

- My model has two factors:
  - Classification by (hour, weekday) combination – eg, sorting into 24 x 7 = 168 bins.
    - My base estimate is the average from this bin.
  - I multiply this by a factor *p* estimating user growth. I calculate this:
    - Using *p* = (checkins in last 24 hours) / (checkins in a typical previous 24 hours) when this information is available
    - Using a logistic regression to estimate the weekly growth rate and estimating *p* accordingly, when the prior information is not available.

Analytics & Modeling

- Relevant file: *uber_analytics_code.py, uber_record.xlsx*
- Cmd line: *python uber_analytics_code.py > uber_record.csv*

Model Creation

In considering the data set, my first rough, intuitive assumption was that this data was made up of 168 different Poisson distributions – one for each weekday/hour combination (24 x 7).

My second is that there might be some “busy-ness” factor b, such that:

- $b_0 = 1$
- b_t is a random walk.
- The mean of *b* follows growth of usage/users (if there is no growth then there is no expected drift)
- For any given one-hour time period *t*, $\lambda_t = b_t * f(\text{weekday}(t), \text{hour}(t))$ where *f* is the aforementioned function

Model Testing

I decided to test the model against the data, generating a csv of all logins binned by hour, with the weekday and hour tagged. (see the XLS).

Tab 1 is the raw output.

In the *Poisson Verification* tab, for each hour I compiled the mean and variance of all hours up to that point with the same weekday and hour, eg for the hour beginning at 11am on 3-17, this would include:

- 11am 3-3
- 11am 3-10
- 11am 3-17

(Since this was an exercise in verification rather than prediction I included the last observation). We’ll call this the *Observable Time Data (Inclusionary)*

I won’t give an overview of the Poisson distribution, as I assume the reader is familiar with it, but every distribution has a key parameter λ , which is equal to both the mean μ and the variance σ^2.

I decided to test the following two question:

- How similar is our sample μ to our sample σ^2?
- What % of our sample s^2 fall into a 90% CI for our sample mean $\bar{x}$?
    - Since our variance is also a sample, without a deep statistical probe of this question, if this is indeed a Poisson distribution I would assume it would be ~81% (=90% * 90%).

The answers are in the *Poisson Verification* tab, P4:Q14.

Some observations:

- As we see in P4:Q12, the average $s^2 / \bar{x}$ is around 1.45 for the data set.
  - Intuitively, this seems close enough to 1 – especially for the compact Observable Data Sets associated with the earlier dates -- to uphold the idea that there is a basic Poisson-ness.
- However, note that this metric seemed to be larger for the larger data sets.
  - Each Observable Time Data associated with the last week of data had 9 observations (though it is somewhat misleading labeled 'average 8'), spanning the whole two months.
- There is some drift in λ.
  - Point 1
    - When $s^2 / \bar{x} = 2.1$ for data sets spanning 2 months (Q12)
    - but $s^2 / \bar{x} = 0.9$ for data sets spanning one week (Q4),
  - This is an indication that there is some change in the underlying parameter λ.

I chose to account for the underlying drift using the formula:

- $x_t = p_{h,\,t}$ * Poisson ($\lambda = \bar{x}$ for x in the *OTD (Exclusionary)*)
- Where $p_{h,\,t} = \frac{\text{avg \# of checkins in the } h \text{ hours before each observation in the } OTD}{\text{\# of checkins in the } h \text{ hours before current observation } x_t}$
- Using $h = 24$

(see columns *D, E* in the *Averages* tab of the Excel workbook and the *secondObjectLast24Hours* and *firstObjectLast24Hours* variables in the *uber_data_analysis.py* script)

With the Poisson distribution this gives:

- $E(x_t) = p_{h,\,t} * \bar{x}$ for x in the *OTD (Exclusionary)*

On several metrics, this is an optimal parameter. See in the *Averages* tab:

- Without regressing: Square error, absolute error, and error as % of predicted value. (Columns W through Y, AA through AC, AI through AK). The adjusted value reduces error by about 10%.
- Using regression (eg comparing, $E(x_t) = m * p_{h,\,t} * \bar{x} + b$ to $E(x_t) = m\,\bar{x} + b$) --- the chart in column *F* of *Averages* illustrates an R^2 of 0.76 for the modified metric vs. 0.72 for the original metric.
- Seeing the positive intercept in the chart for the original metric brings up the question: are $E(x_t) = p_{h,\,t} * \bar{x}$ or $E(x_t) = \bar{x}$ unbiased estimators? Breaking down the observations by cohort week, graph in column *BC* shows that while the modified estimate account is unbiased and in the aggregate accurate (with an actual : expected ratio near 1), the original estimate is much less so.

Model Adjustment Based On Available Data

However, the scenario given here calls for making a battleplan rather than being able to adjust on the fly as new info comes in.

The best way to model the drift over time would be as an exponential function, since user growth over time is exponential. I do this in the chart in column BJ, being careful to constrain b = 1 (since growth should start from the starting point!). We see there is 2.9% weekly growth (BH4).

I model this growth in the app, calling the estimated slope *m*

API

- Relevant file: *uber_api_code.py*

API Operation

- The API takes in the following URLS:
  - /api/post
    - POST: This is how you upload JSON
  - /api/data_analytics
    - GET: Returns predictions as a JSON
  - /api/all_data
    - GET: Returns all dates as a JSON
    - DELETE: Deletes all dates
  - /api/all_data_binned
    - GET: Returns the binned JSON
- The relevant commands are:
  - curl -i -H -d "Content-Type: application/json" -X POST -d @uber_demand_prediction_challenge.json http://localhost:5000/api/post
  - curl -i http://localhost:5000/api/data_analytics > output.txt
  - curl -i -X DELETE http://localhost:5000/api/all_data

API Design

- Basic components:
  - The API I built has three basic components:
  - **times –** the set of all logins, ordered by the time they were imported to server
  - **two_category_buckets** – the set of all logins, stored as a dictionary.
    - Each keys is a 2-tuple (hour, weekday).
    - Its corresponding values is an array of all logins with that constraint
    - Taking the length of any value array gives the total # of logins with that hour and weekday.
  - **hourly_bucket** - the set of all times, ordered by date.
    - The index corresponds to "hours since Jan. 1, 2006"
      - As a result, the list has many blank and empty element

            - This design choice was made in order to handle the use-case of dates being added prior to the earliest existing date, necessitating a massive, painful list re-indexing…
    - **prediction** – an array listing the times that need to be predicted for.
- Design:
    - Construction of *times* was attached to the POST module.
    - Construction of *prediction* was attached to the GET module at /api/data_analytics
    - Construction of the *two_category_buckets* and *hourly bucket* were, while important for analytics processing, attached to the POST module because they change the data structure, so it doesn't make sense for them to be associated with a GET request. This initial setup cost is well worth the time in processing cost later.
- Scaling
    - As data sets get larger, it may make sense to store this data in a database – the event data is a good candidate for SQL SELECT WITH clauses. Right now, the program will run in memory, so this is faster.
    - The
- Drawbacks:
    - The main design challenges are this API involve repeated checking of whether a list index:
        - Is in the array range so can be referenced; if not, must be appended.
        - Is in the array range, but has an empty value which must be initialized to *[element]*
        - Is in the array range, and has a non-empty value which can simply be appended to.
        - *Ugh.*
- Statistics // Model:
    - The current model implemented in lines 222-274 of code is to:
        - use the modified expected value (MEV) for a Poisson distribution, when relevant 24-hour-prior information is available
        - or use the exponential model, for when it is not.
    - In math:
        - $E(\text{logins}_t) = p*E(\lambda) = p*\bar{x} = p*\frac{\text{(total \# of obs in all periods with same (hour,weekday))}}{\text{(total \# of periods with same (hour,weekday))}}$
        - Where $p = \begin{cases} \frac{logins\ in\ last\ 24\ hours}{logins\ in\ avg\ 24\ hours\ preceding\ (wkday,hr)}, & \boldsymbol{data\ is\ available} \\ \frac{g_t}{\bar{g}\ \boldsymbol{for\ g\ in\ the\ training\ set}}, & \boldsymbol{data\ not\ available} \end{cases}$
        - $g_t = m^t$ (*m* to the power *t*); m is our estimated slope, ie, weekly usage growth
        - *t* = time in weeks since the experiment began.
    - Note that the first case (MEV) only occurs for the first value (05-01-2012 00:00) in our test set, because after that the "last 24 hours" data isn't fully available.
        - Were this an actual system, it would be updating with real-time data being POST-ed and be able to invoke MEV more often.

- Edge Cases
    - The prevIndex code in *bin_dates* eliminates the problems of hours with no logins associated, by creating empty hours in *hourly_bucket* where they are found.
    - One edge case I have not yet sufficiently addressed involves where only a few dates are inputted, insufficient to generate 24x7 = 168 keys in two_category_buckets.
    - The program currently assumes that all dates entered into the program are contiguous – if you feed it logins from Jan to March 2011 and Jan to March 2012, it will think you are feeding 15 months of data instead of 6 and divide accordingly.
    - The program needs at least a week of login data, otherwise it will possibly return divide by 0 exceptions or inaccurate information when running the relevant regressions. I believe I've solved this using a *try* loop, but I don't have time to be sure.