2014 CCP: Data Scientist
Challenge 3

Candidate: Giuliano Janson

General approach

I utilized the Python Anaconda distribution for all 3 parts of the challenge on a W7 machine with 16GB of RAM and an i7 quad core processor. Anaconda come pre-loaded with some of the most utilized scientific packages for Python. In particular I used scikit-learn for machine learning, Pandas for data manipulation and networkX for graphs analysis.

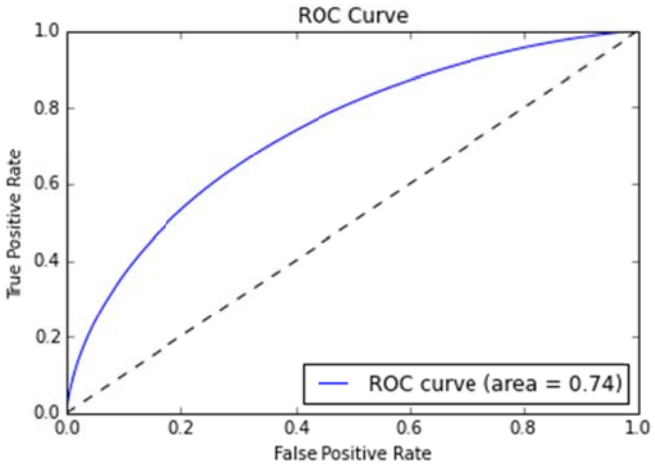
Part 1 - SMARTFLY

Approach: The SMARTFLY problem is essentially a pure machine learning problem. With AUC being the known evaluation metric I sought the best algorithm to maximize that metric on a 20% validation set.
Given the presence of many categorical features, and the likely interactions of these, I decided to use a model that can deal effectively with both categorical and numerical features, as well as missing values, out of the box. Random Forests seem a great fit because require pretty much no data preprocessing (i.e. no one-hot-encoding required), are great at modelling interactions, and can deal with missing data. The randomness of Forest models also works exceptionally well in dealing with feature correlation, another likely problem with this data.

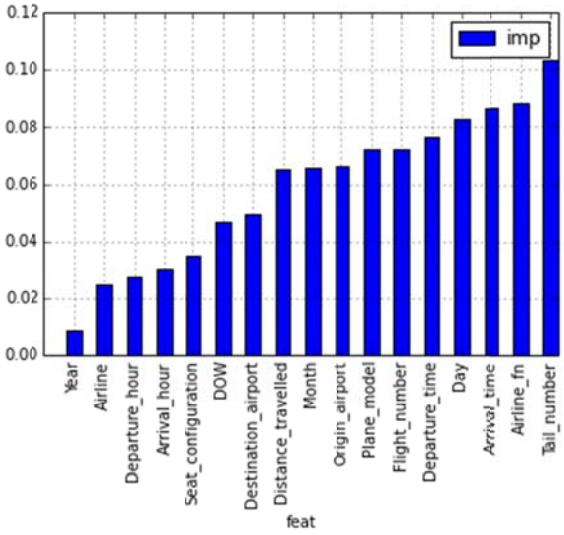
Feature Engineering: the dataset was already feature reach so I limited feature engineering to transforming time features (Expected Time Arrival/Departure) into numerical sequences. I created a couple of other features such as month, day, day of week.
Then I encoded categorical feature into numerical because scikit-learn requires numerical matrices as inputs.

Model: I tried different algorithms, including Gradient Boosting, Logistic Regression and linear SVM. Random Forest seemed to provide the best performance. After some grid search over a set of hyper-parameters a Random Forest with 200 trees and a minimum number of observation per leaf equal to 10 (to improve generalization) seemed to perform best.

Random Forest - Validation ROC



Random Forest – Feature Importance



Validation: I tried both random splits and time dependent splits, both returned similar results. My best Random Forest model ended up producing a .74 AUC on a 20% validation set.

Other things I tried: I tried a few other things that resulted in no/little improvement over the basic Random Forest model. These include:

- Using the continuous variable Delay as a weight to train my models (this did not improve performance)
- Using logistic Regression and linear SVM on one-hot-encoded features (this performed significantly worse than the Random Forest)
- Using and ensemble of Random Forest and Logistic Regression.

Time spent: ~8 hours

Part 2 – ALMOST FAMOUS

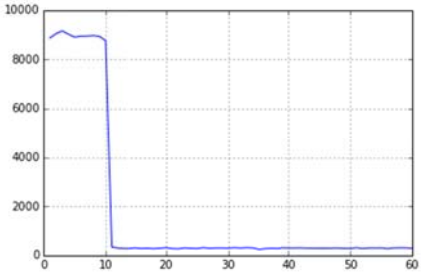
Approach: There were two main things to be decided from an approach perspective in this problem.

- How to identify bot activity
- How often to evaluate G and Z test outcomes (the last two questions).

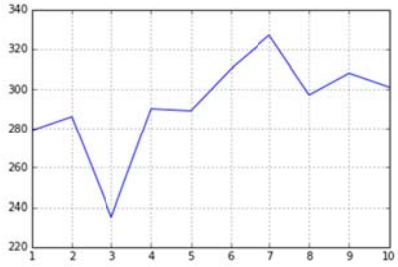
Problem 1 - BOT identification: Essentially all that the bot logs provide is time stamps and actions for each visit. As such, it seems intuitive that BOT identification has have something to do with time-to-ad-click. I decided to use the time between landing on a page and ad click as a feature to differentiate bot activity from legitimate activity.

In fact, it turns out that all bots on the spam file will click on an ad within 10 seconds from landing on a page. On the other side it is safe to assume that visitors who signed up or purchased something should be legitimate activity. None of these visitors clicked on ads within 10 seconds. The charts below clearly show that 10 seconds appears to be a good threshold to identify bots. Therefore I identified and removed from the logs file each and every user id where existence of such a pattern existed.

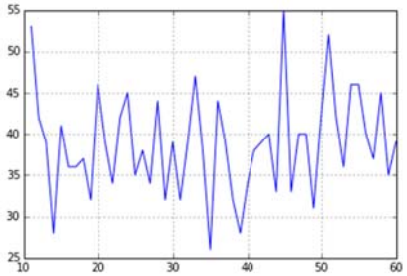
all logs



bots



legit



Time between landing and first ad click

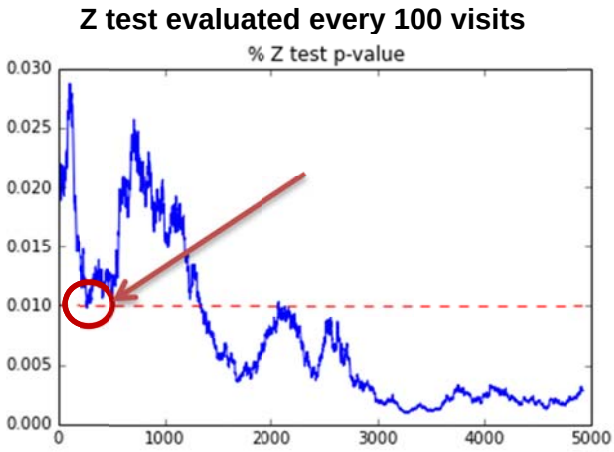
Problem 2 & 3: these two problems ask straightforward questions. All my calculations are documented in the code I have provided.

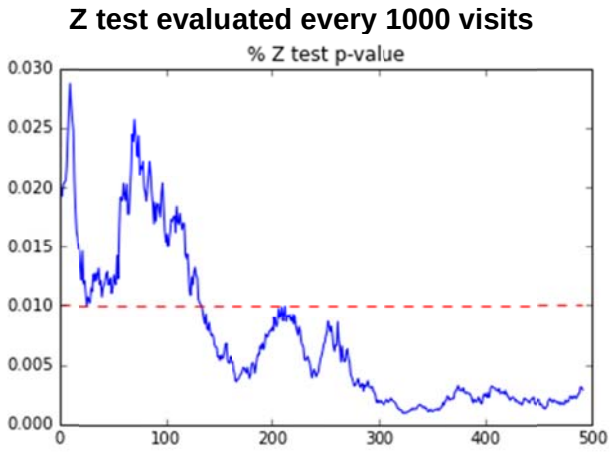
Problem 4 & 5: these questions are straightforward in how G and Z test need to be performed. The most critical choice to be made here is how often to evaluate the outcome of these test. One could run these test every time a new visit is recorded, or on any chosen time interval (minute, hour, day,…).
While the data is provided retrospectively, the real value of determining the earliest point the experiment showed statistical significance lies in the ability to stop a live experiment at that time. An underlying problem exists regarding “peeking at the experiment” while the experiment hasn’t concluded yet. There are pros (mostly financial) and cons (mostly lack of accuracy of reported significance) in peeking and interrupting an experiment before what should be a pre-determined end (and thus a pre-determined duration).
In this very case this isn’t an actual problem because the experiment has likely been carried on to a pre-determined end and all the data is provided retrospectively. However I believe that the solution should reflect as solid of an approach as if this were a live experiment. With those methodological concerns in mind, I decided to evaluate the experiment as follows:

- Every day for Problem 4.
- Every 1000 visits for Problem 5.

My results follow this methodological choice, but different results could follow other evaluation intervals. In particular for the revenue experiment, which has much higher variability due to the different sources of revenue being on such a different magnitude, it is not unlikely to get earlier significance due to chance.

For example, the two following charts show Z test p-values from date 9/26 forward. The first chart records accumulated Z test every 100 visits and has one isolated occurrence of statistical significance one day before the p-values eventually settles to significance. This outlier could be a function of how I'm aggregating the data, nevertheless it highlights the importance of setting a solid evaluation interval.





Time spent: ~8 hours

Part 3 - WINKLR

Approach: Network analytics was something new to me prior to this challenge so I started with a literature review of common analytical approaches to link prediction. My initial idea was to turn the given problem into a supervised problem by randomly removing links from the existing graph and then using graph features (such as proximity measures) and a supervised algorithm to measure performance. However the data that needs to be predicted seems to be a very specific subset of all missing links (these are "users who click frequently on other users they currently do not follow") and thus one has to wonder how well a model built on the entire graph would generalize to such a specific subset of it. Especially because "click frequency" is not a feature available for the entire graph. The concern here is that, given an evaluation metric, this approach would result in a cross validated metric value that is way off from what the test will result in, due to the test set not being representative of the whole network.

In light of that, I decided to take a simple approach based on shortest path that is known to work reasonably well for most problems. I calculated shortest path for all tuples in in the test set, sorted by shortest path ascending, and further sorted by average shortest path of a potential follower's neighbors to a potential followed user.

Time spent: ~8 hours