##### **Solution #6 overview**

P

237

First of all, all those who managed to get decent submissions out of this huge dataset deserve kudos. Even if you score is below Kirk's shared kernel. What you did is way more valuable, and lessons learned here will help you later.

Second, sharing is great when done in good faith, and lots of people did share a lot here, too many of them to name them. Eve people who started here, like [@Samrat](https://www.kaggle.com/Samrat), shared a lot. This is what makes this community so valuable.

Third, thanks to [@inversion](https://www.kaggle.com/inversion), Kaggle, and Talking data for organizing a very challenging, ans almost leak free competition. I say almost because it is clear now that the test data was sorted by click time then target value. Exploiting this was the final twist that helped some of us fare better. But the impact is not that large, I estimate it to be about 0.0004 for me. And it was [disclosed soon enough](https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/discussion/55677) for everyone to react to it. Thanks to [@plantsgo](https://www.kaggle.com/plantsgo) for sharing it soon enough.

I didn't decide to go solo from the start, but as time went by, I saw I was making progress every day, and decided to go solo till the end. In retrospect I am not sure it was wise, I didn't sleep much in the last week ;) I did receive some invites to merge during the last week before deadline, and I thank people for them. I did miss a very late invite from a top 10 team as I was away that evening. I wonder what would have been our score if we had teamed.

Anyway here is my solution. Given I was solo, and given the size of the data set, which meant hours to produce a submission, I decided to focus. I focused on a single type of model, LightGBM. I split my time roughly as follows:

* 80% feature engineering
* 10% making local validation as fast as possible
* 5% hyper parameter tuning
* 5% ensembling

I spent most of my time doing feature selection, as my machine was not usable with 50 features or more. I wish I had used the [trick shared by Kruegger](https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/discussion/56105), maybe I would have been able to add more features. However, being forced to be selective about features probably led to better models in the end. And I would not have been able to move past40 features without using the 'two\_round' parameter as suggested by [@authman](https://www.kaggle.com/authman).

My private LB score comes from a single lgb run with 48 features that scored 0.9825 public and 0.9835 private. I submitted a blend of this with 5 other similar models that yield 0.9828 public and 0.9837 private, but for some reason [that sub isn't taken into account](https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/discussion/56234). This is OK as my rank would not change with it. I didn't submit all the 5 models individually during the competition, but did it after to get their score. The best single lgb run scores 0.9827 public and 0.9836 private, with 48 features.

I mostly used a 20 core Xeon at 2.3 MHz with 64 GB RAM and 64GB swap. That machine is a bit slow, but it scales when using 20 threads. I also used another machine with a 4 core i7 and 2 GPU to run Keras (see below).

****Validation****

Let's us look at validation. This is key. If you don't have a good validation scheme then you rely solely on LB probing, which can easily lead to overfit. I ended up settling on:

* training on day <= 8, and validating on both day 9 - hour 4, and day-9, hours 5, 9, 10, 13, 14.
* retraining on all data using 1.2 times the number of trees found by early stopping in validation

Using two validation sets was to make sure I was not overfiting to one of them. The hours were selected to match the public and private test data. I also watched the train auc in the early days, discarding features that improved validation but also increased the gap with train a lot. I stopped watching train auc in the last week to speed up things, but last time I checked I had a quite small gap.

I also used LB, i.e. only kept a feature if local auc and LB improved. Yes, I know this can lead to overfit, but given I was filtering first on local validation I think I escaped it for the most part.

This was a very effective scheme, with the hour 4 score being the same as LB score with a std difference around 0.0001. However, it is very time consuming because of the computation of the auc metric for early stopping. In order to speed it for feature evaluation I used two lighter ways. First, using only day 9 data, with 5% of hour 4 data for validation, the rest for training. This could run in less than one hour, and was used as a filter. Only features that improved on that went to the next stage which was train on day 8 and validate on day 9, both hour 4 and other test hours. This was also a very effective scheme, with very good correlation with LB score, but it was not effective when evaluating lag features. I therefore switched to training on day <= 8 later on.

Another way of speeding feature evaluation was to share each feature in a separate feather file. This way, testing a feature set only requires assembling a set of files into one dataset. Features were mostly tested by adding them one by one, and keeping them if local validation score improved by at least 0.00005. I also added several of them at once, then removed them one by one to see if validation score decreased. I basically did feature selection full time for the competition, preparing experiments to be run while I was away during day, or while I was sleeping. The machine never stopped.

****Feature Engineering****

Features were computed on the concatenation of train and test\_supplement, sorted by click time then by original order. Now I am not sure the second item was useful.

I used several families of features.

* Only app, and os from the original features were kept. They were handled as categorical, and were my strongest 2 features with a third category made of the hour in the day.
* China days. Introduced 24 periods that start at 4 pm. These were  
  used for lag features based on previous day(s) data.
* User: ip, device, os triplets.
* Aggregates on various feature groups, similar to what was shared in many public kernels. Aggregates I used were count, count of unique values, delta with previous value, delta with next value. Time to next click when grouped by user was important. Other useful ones I didn't see in kernels: delta with previous app.
* Lag features, based on previous China days values. Previous count by some grouping, and previous target mean by some grouping. The latter was a weighted average with the overall target mean, the weights being such that groups with few rows in it had a value closer to the overall average. This is a standard normalization in target encoding.
* Ratios like number of clicks per ip, app to number of click per app.
* Not last. This was to capture the leak. It is one except for rows that are not the last of their group when grouped by user, app, and click time. I ignored channel as I think that clicks are attributed to the most recent click having same user and app as the download.
* Target. This is to also capture the leak. I modified the target in train data by sorting is\_attributed within group by user, app, and click time. The combination of both ways to capture the leak led to a boost between 0.0004 and 0.0005.
* Matrix factorization. This was to capture the similarity between users and app. I use several of them. They all start with the same approach; construct a matrix with log of click counts. I used: ip x app, user x app, and os x device x app. These matrices are extremely sparse (most values are 0). For the first two I used truncated svd from sklearn, which gives me latent vectors (embeddings) for ip and user. For the last one, given there are 3 factors, I implemented libfm in Keras and used the embeddings it computes. I used between 3 and 5 latent factors. All in all, these embeddings gave me a boost over 0.0010. I think this is what led me in top 10. I got some variety of models by varying which embeddings I was using.

****Hyper parameter tuning****

I spent time given how long it is to run an experiment, but I didn't tune much. Main settings were to scale positive by around 400, use an initial score that minimizes expected loss if target is constant, and min child per leaf to be such that it requires at least 1 positive plus another example, in order to avoid overfiting to single positive examples. I used 31 leaves and a depth of 8.

****Ensembling****

On my local validation, the best way to blend several models was to average the logit of the predictions (aka raw predictions). I started doing restacking, i.e. adding validation predictions to day 9 features, and training on it, but this was hitting my 50 feature limit, and runs were very long. I did not ran it the last day for that reason. It may have given me a little additional boost, but I don't think it would have been enough to move up in the LB, because my models were not diverse enough anyway. I also think that using only day 9 for second level was leaving too much on the table. I thought of generating of prediction for the full dataset, but that was a daunting task. I now see this is what [@bestfitting](https://www.kaggle.com/bestfitting) did, great move on his part.

****Takeaway****

I think it was crazy to do this solo, too much work for a single person. I really admire the other fools that went same way. Once solo, I am not sure I should have done things differently, except for spending time to alleviate the 50 features limit.

Also, as often in my competitions, I make a lot of progress the last day, not sure why. In this case I moved from 0.9824 to 0.9828 public, and 0.9832 to 0.9837 private. The lesson is to never give up, and not let the public LB dictate your mood.

I hope the above will be useful to some. Thanks for reading it all ;)

Edit: I [shared my libFM implementation](https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/discussion/56497).