# Findings and Justifications for the OXL Problem “Ads Recommendations” on Hackerrank by Lasse Deleuran (LasseD)

This challenge is most likely meant to be solved using machine learning algorithms. This is heavily implied by the problem statement, pre-test tasks and recommended reading. I have, however, taken a heuristic approach to this challenge as my experience tells me that you can get some fun out of making good guesses. In this document I will answer the points asked for this challenge.

## Review of the datasets

This problem provides 4 datasets: “user\_data.csv”, “ads\_data.csv”, “user\_messages.csv” and “user\_messages\_test.csv”:

### user\_data.csv

This file contains 2.143.889 rows with information for 12.365 unique users. Some of the columns are sparsely documented in the problem setting. The columns worth mentioning in this context are:

* "event\_time": This is an actual time with year, month, day, hour, minute and seconds rather than milliseconds as stated in the problem statement. The data is not sorted by event\_time, so one of the first steps in my algorithm is to sort by and easily find the “last X impressions”.
* "origin": This column can contain the following values: "browse", "browse\_search", "home", "notification\_center", "search", "push", "drawer" "" (the empty string) and "deeplink". While the meaning of most of these can be guessed, it should have been documented better.
* "images\_count": While it makes sense not to show ads without images (research say they do poorly).
* "ad\_impressions", "ad\_views" and "ad\_messages". These columns are sparsely documented. They are shown for the given timestamp and can be used to get an idea of how popular the ads become. Unfortunately, I didn’t have the time to use this insight.

### ads\_data.csv

This file contains data for 645.168 ads. There has been an error in the creation of this document so the special characters, comma, and quotation mark, have not been escaped. Loading this data into my software thus required verification that ‘”,”’ is used as cell delimiters around the fields with text, “title” and “description”. The columns in this file are as expected and I feel the documentation for them is better than for user\_data.csv.

The quality of this dataset is very high as all referenced ads in user\_data.csv and user\_messages.csv are present.

### user\_messages.csv and user\_messages\_test.csv

user\_messages.csv contains 10348 lines with recommended ads for user/category pairs. This file is supposed to be the training file, so the challenge can be formulated as: See what OXL recommends using this file and make your own recommendations for the user/category pairs in user\_messages\_test.csv. Notice. There are users in both files which are not present in user\_data.csv! These files are fairly simple, but it’s worth noting that there are rarely more than two recommendations for a pair in user\_messages.csv, while the problem statement ask for up to 10 suggested pairs in ads\_recommendations.csv. The Hackerrank users who run machine learning algorithms would benefit from less sparse training data.

## The Data Preprocessing Steps

Since I’m running heuristics rather than machine learning algorithms, the data preprocessing steps are simple. I load the data from user\_data.csv, ads\_data.csv and user\_messages.csv into memory and make some very simple lookups:

* User data is stored in structs of the type “User” where impressions (the data from user\_data.csv) are stored sorted by the timestamps. For each impression I only store the ID of the ad and the timestamp. Finally, the maximal squared distance among the last 5 impressions is stored in the variable “maxViewDist”. It is used to determine if an ad lies close to the last place the user was “seen” and thus whether it should be recommended or now.
* All ads are similarly sparsely stored: Only the ID (“ad\_id” from ads\_data.csv), position (“lat” and “long” from ads\_data.csv) and “enabled” is used by my simple heuristics. The ads are stored in the “Category” struct and divided by their category.
* For each of the 10 categories (362, 800, 806, 811, 815, 853, 859, 881, 887 and 888) the “Category” struct further stores the ads ordered by how many times they are recommended to users according to user\_messages.csv (in the vector topRecAds) as well as the ads ordered by how many times they are present in lines from user\_data.csv (in the vector topImpAds).

All these data structured are created in the first part of the main-method, i.e. before line 392.

### Model Used for the Recommender System

The heuristic used for recommending ads is simple: I recommend 10 enabled ads for the given category after the following criteria:

* If a user isn’t present in user\_messages.csv, then the most highly recommended ads in user\_messages.csv are recommended. They are ordered by how many times they are present in user\_messages.csv. If there are not 10 of these, then most highly viewed ads from user\_data.csv are added until there are 10 in total.
* If a user is present in user\_messages.csv, then I simply recommend the last 10 viewed ads from user\_messages.csv. Again. If there are not 10 of these, then I add highly recommended ads from user\_messages.csv and user\_data.csv. It has turned out that I could fine tune the recommended ads by considering the position. First I only consider ads from user\_messages.csv whose square distance form the last position of the user is within 0.01 of “maxViewDist”. If this does not result in 10 recommendations, then I repeat for “0.1” rather than “0.01”, then for ads in user\_data.csv and finally I skip this location restriction to ensure there are 10 recommended ads.

### Further considerations

The problem statement mentions that precision is very important for the scoring, and I see that the order of the recommended are significant. I have tried to consider precision in my recommender, but found that always outputting 10 ads yields a superior score compared to outputting fewer recommended ads.

I have also tried more advanced recommender, such as finding ads with matching keywords, ads recommended by users with similar patterns in user\_data.csv and even an Amazon-like “others who viewed this ad also viewed” recommender (see oxl\_raw.cpp). All these attempts gave poorer results than the simple heuristic and were thus scrapped.