Evaluation of Cohort Algorithms for the FLoC API

Google Research & Ads[1]

ABSTRACT

The Federated Learning of Cohorts (FLoC) API is a privacy preserving mechanism proposed within the Chrome Privacy Sandbox, for enabling interest based advertising. The API is based on the notion of cohorts - groups of users with similar interests. In this paper we evaluate different methods for generating cohorts, showing clear trade-offs between privacy and utility. Using proprietary conversion data, we demonstrate that generating cohorts based on common interests can significantly improve quality over random user groupings. In fact, we achieve a 350% improvement in recall and 70% improvement in precision at very high anonymity levels compared to random user grouping.

Introduction

On January 14, 2020 Chrome published an intention to implement a Federated Learning of Cohorts (FLoC) API. The goal of the FLoC API is to preserve interest based advertising, but to do so in a privacy-preserving manner. More precisely, the FLoC API relies on a cohort assignment algorithm. That is, a function that allocates a cohort id to a user based on their browsing history. To ensure privacy, Chrome requires this cohort id to be shared by at least k distinct users.

The goal of this paper is to share some initial results about the efficacy of certain algorithms that would abide with the FLoC principles and begin a discussion about different ways for implementing and evaluating cohort assignment algorithms. We believe in the long run ad tech providers could use cohort ids (potentially paired with TURTLEDOVE) as a feature in their ads personalization algorithms.

We use the following higher level principles largely laid out for FLOC within the Privacy Sandbox to design some experimental cohort assignment algorithms

1. The cohort id should prevent individual cross-site tracking.
2. A cohort should consist of users with similar browsing behavior.
3. Cohort assignments should be unsupervised algorithms, since each provider has their own optimization function.

[1] Primary contacts: Deepak Ravichandran & Sergei Vassilvitskii

4. A cohort assignment algorithm should limit the use of “magic numbers”. That is, its parameter choice should be clearly and easily explained.
5. Computing an individual’s cohort should be simple. This allows for it to be implemented in a browser with low system requirements.

The particular algorithm design tested here proposes hashing the browsing history of a user into a p-dimensional binary vector using the SimHash algorithm and defines a cohort to be all users sharing the same hash. In addition to simulating the proposed FLoC design we evaluate other plausible clustering algorithms and compare them based on the following three dimensions:

- **Privacy**: What fraction of users are in large cohorts?
- **Utility**. What is the similarity of users in the same cohort?
- **Centralization**. Does information need to be sent to a centralized server to calculate a cohort id?

Privacy vs Utility

At a very high level, a cohort assignment algorithm presents us with a privacy-utility trade-off: the more users share a cohort id, the harder it is to use this signal to derive individual user’s behavior from across the web. On the other hand, a large cohort is more likely to have a diverse set of users, thus making it harder to use this information for fine-grained ads personalization purposes. An ideal cohort assignment is one that generates cohorts by grouping together a large number of users interested in similar things (see Figure 1).

Measuring Privacy

One privacy requirement for the FLoC API laid out in the Privacy Sandbox is that each cohort is k-anonymous. We say a cohort id is k-anonymous if it is shared by at least k users. The higher the value of k, the more privacy protection we provide to a user. K-anonymity allows a user to “hide in the crowd”, making derivation of individual behavior across the web harder.

We want to emphasize that, even though differential privacy is now the de facto privacy notion in industry and academia, we decided against using it as our privacy measure for building audiences. This choice was made since we believe it fails to quantify the hardness of tracking users across the web. In fact, a cohort id can be differentially private and still be used as a fingerprint. Indeed, an algorithm that assigns a unique, random (but fixed across websites) cohort id to each user will satisfy the definition of differential privacy; since the random id does not reveal any user information. Nevertheless, it is clear that this unique, random id can be used as a stable fingerprint with which to track users across the web. On the other hand, a k-anonymous cohort id cannot, by definition[2], be used as a fingerprint.

[2] if used in totality (ie not paired with other signals)

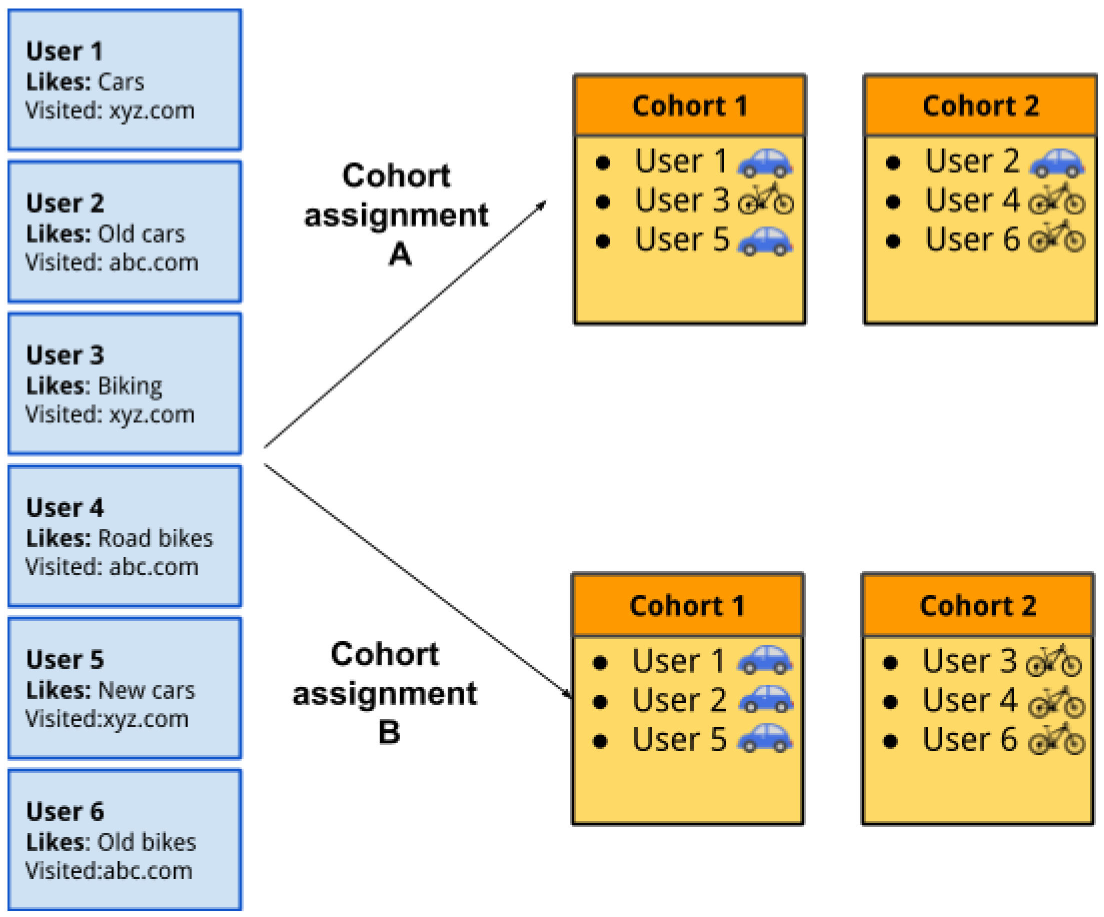


Figure 1. An example of how two different cohort assignments with the same anonymity properties can yield different utility results. Here we show six users split into cohorts in two different ways. Assignment A creates a cohort of xyz.com visitors and another of abc.com visitors. Assignment B generates a cohort of car enthusiasts and one of bike enthusiasts. Even though both achieve k-anonymity for k=3. Assignment B is more likely to be effective for interest based advertising.

Algorithm descriptions

We now introduce the algorithms we will be using to calculate cohort ids. The computation of cohort ids is fundamentally a clustering operation: grouping similar users together. The input to the clustering algorithm is a set of vectors, one vector per user, in a d-dimensional space. We will later discuss how to generate these vectors based on browsing history.

SimHash

SimHash is an instantiation of the popular locality sensitive hashing (LSH) family of algorithms. Initially developed with the goal of identifying near duplicate documents quickly, SimHash takes as input a d-dimensional vector x and outputs a p-bit vector $H_p(x)$ which we refer to as the hash of x.

The i-th coordinate of the hash vector is obtained by the following rule:

$$H_p(x)_i = 0 \; if \; w_i \cdot x \leq 0 \qquad H_p(x)_i = 1 \; if \; w_i \cdot x > 0 \; ,$$

where $w_1, \ldots, w_p$ are random unit-norm vectors. A cohort corresponds to all users whose input vectors share the same hash. Figure 2 shows an example of the SimHash function H_3.

SimHash has the property that similar vectors are more likely to be hashed to the same cohort id than dissimilar vectors. More precisely, if x_1 and x_2 are two vectors, then the probability of mapping x_1 and x_2 to the same cohort id is given by:

$$Prob(H_p(x_1) = H_p(x_2)) = (1 - \frac{\theta(x_1, x_2)}{\pi})^p \; ,$$

where $\theta(x_1, x_2)$ corresponds to the angle between vectors x_1 and x_2. That is, input vectors with small angles between them are exponentially more likely to share the same hash than input vectors with a large angle between them. Alternatively, vectors with high cosine similarity[3] are more likely to be in the same cohort.

The main advantage of using SimHash is that the computation of the cohort id for one user does not depend on the information of others. Given a vector x, the cohort id can be calculated in the client without knowledge of any other user's information. The properties of SimHash ensure that the cohort id calculated in this manner is shared with users that have similar input vectors. In particular, there is no need for any centralized data collection to compute cohort ids. In spite of this, the properties of the SimHash algorithm will ensure that cohorts generated in this way will consist of similar users. This is a remarkable feature of the SimHash algorithm as it allows for clustering to happen without a central server ever storing a user's browsing history.

The main downside of SimHash is that a minimum cluster size cannot be enforced. Nonetheless, this problem can be solved by having an *anonymity server* that tracks the size of each cohort. This server could block the API from returning a cohort id if the cohort is not k-anonymous. Since the server only gets to access a small bit length hash of a user's browsing history, the amount of information revealed to the server is minimal.

[3] The cosine similarity S between two vectors is defined as their normalized dot product. One can recover the angle between two vectors by taking the arc cosine of S.

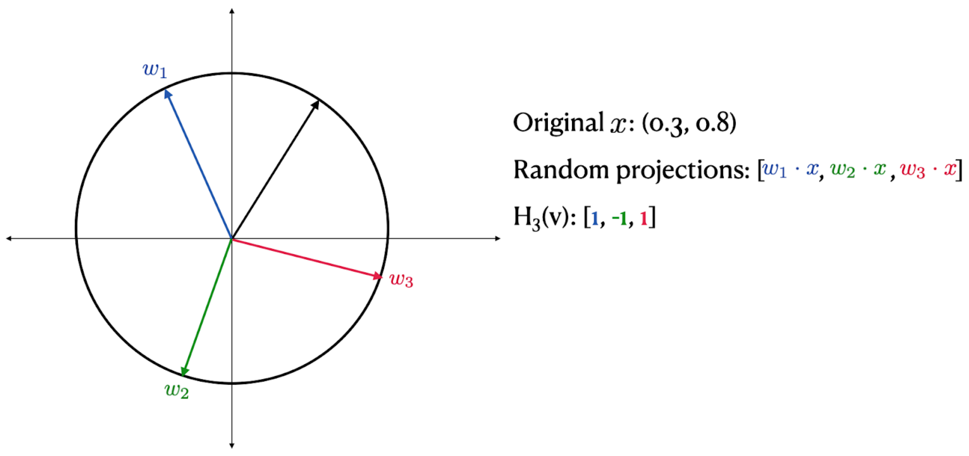

w_1
w_2
w_3
Original x: (0.3, 0.8)
Random projections: $[w_1 \cdot x, w_2 \cdot x, w_3 \cdot x]$
$H_3(v)$: [1, -1, 1]

Fig 2. Example of a 3-bit SimHash calculation.

SortingLSH

The choice of the number of bits p defining the SimHash algorithm is crucial. If it is too low, cohorts will be large, making it more likely for dissimilar users to be part of the same cohort. On the other hand a large value of p can result in cohort ids that are shared by only a small number of users, violating the k-anonymity requirement. Moreover, for some datasets, we have observed that the cohorts generated by SimHash have very heterogeneous sizes: a few very large cohorts and a large number of small cohorts. In this scenario, “splitting” the big cohorts by increasing the number of bits for SimHash is impossible as it would break the k-anonymity of smaller cohorts (see Figure 3).

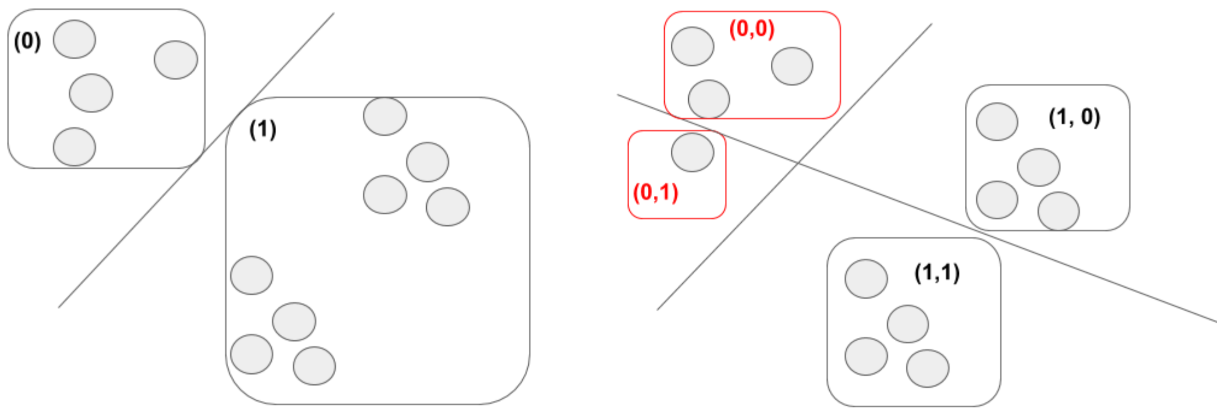


Figure 3. Example where we want to enforce k-anonymity for k=4. On the left, using 1 bit of SimHash generates k-anonymous cohorts, but cohort (1) is very big and not homogenous. On the right, cohort (1) is “split” into cohort (1,0) and

cohort (1,1) by using a 2-bit SimHash. This, however, results in cohorts (0,0) and (0,1) violating the k-anonymity restriction.

SortingLSH is a method that solves this issue and ensures k-anonymity while improving the quality of SimHash at the same time. This is achieved by homogenizing the size of cohorts. SortingLSH is a minimally centralized method that acts by post-processing SimHash clusters to ensure k-anonymity.

Let $h_1 = H_p(x_1), \ldots, h_n = H_p(x_n)$ denote the p-bit hashes generated by SimHash on all users. Instead of assigning generating cohorts by grouping together users with the same hash, SortingLSH generates cohorts as follows:

1. Sort $h_1, \ldots, h_n$ in lexicographical order to obtain a sorted list of hashes $h_{(1)} \leq \ldots \leq h_{(n)}$
2. Assign the sorted hashes to cohorts by generating contiguous groups of at least k hashes.

The ordering step ensures that contiguous hashes in this order correspond to similar users. The second step ensures that cohorts always have at least k-users.

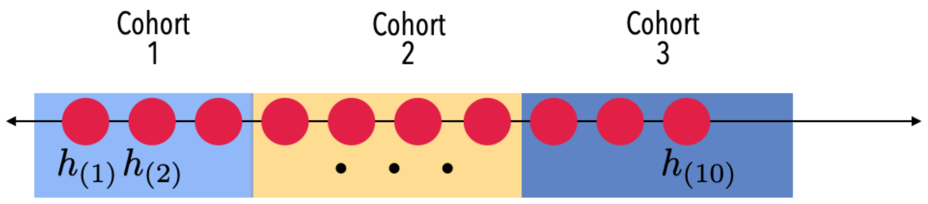


Fig 4. Exemplification of SortingLSH enforcing k-anonymity with k=3. The first cohort consists of the first three sorted hashes, the second the 4th to the 7th and the third cohort is made up of the 8th to the 10th hash. All cohorts have at least 3 users.

The implementation of SortingLSH does require a central server to sort all user hashes and calculate k-anonymous cohorts. However, this information is also needed by a server that enforces k-anonymity on cohorts generated using SimHash. Therefore, the level of centralization is no worse than that of SimHash using an anonymity server.

Affinity hierarchical clustering with centroids

To better understand the privacy-utility trade-off we also consider centralized clustering methods. While a naïve implementation of these algorithms require raw data to be sent to a central server, they can potentially be implemented in the future using Federated Learning technologies. In this section we discuss a variant of the centralized clustering method *Affinity hierarchical clustering* which is based on computing a user-to-user similarity graph.

Affinity hierarchical clustering leverages the structure of a user similarity graph (i.e., a graph where similar users are connected by edges) to inform the creation of clusters. The first step of the method is to compute a graph where users are nodes and edges connect two users if their vectors are similar. Then, the algorithm performs a hierarchical clustering in a bottom-up fashion, creating larger clusters by merging smaller clusters connected by highly similar pairs of users. In this variant of affinity hierarchical clustering, we also control explicitly for minimum size of clusters. Finally, users are assigned to clusters by nearest neighbor search. We discuss these steps in more detail.

Graph construction
We first create a user-to-user similarity weighted graph (i.e. users are nodes, and edges encode similarity). More precisely we create a nearest neighbor graph built over the input vectors associated with the users, using cosine similarity-weighted edges. This is achieved by using efficient locality sensitive hashing techniques to identify pairs of users with high similarity.

Graph clustering
The algorithm proceeds by clustering the users using hierarchical agglomerative clustering. First each node (i.e. a user) in this graph is assigned to a distinct cluster. Then, pairs of clusters are merged until each cluster reaches a specified minimum cluster size. Then, for each such cluster we obtain a centroid representing the cluster by averaging the users profiles in the cluster which is associated to a unique cohort id.

User to cluster assignment
The final step of the method is to associate each user with a cluster. This is simply done by assigning each user to the cohort corresponding to the nearest centroid.

Unlike SimHash or SortingLSH, affinity clustering uses the information of users to actively search for similar users. Therefore we should expect to get a much better privacy-utility trade-off from this algorithm. Nevertheless, notice that to execute the first steps of the algorithm, a server needs access to the raw browsing history of users. As mentioned above, this concern could be assuaged by eventually using federated learning technology. For now, we present the results of this algorithm as a baseline to show a close to optimal utility-privacy trade-off curve.

Public data sets

The algorithms proposed should create relevant user cohorts when applied to any dataset representing users and their interests. Therefore, it is reasonable to start our evaluation by measuring each algorithm's performance on publicly available datasets.

Million Song Dataset
The million song dataset (MSD) is a collection of 1 million songs tagged by categories and user ids. The dataset consists of the listening history of 650 thousand users. Each (user, song) pair is tagged by the number of times it was listened to as well as the categories the song belongs to. These categories have weights representing how well the category describes each song.

Examples of these tags are “Pop” or “60’s”. The dataset also includes some subjective tags such as “good” or “awesome”. The original dataset consists of more than 200 thousand categories. However, most of these categories appear only a few times in the whole dataset. For this reason we restricted the set of categories to those who appear in at least 1% of the songs. Table 2 describes the relevant statistics about the dataset.

MovieLens 25M

A standard dataset for evaluating recommendation systems, the MovieLens 25M dataset consists of ratings of 25M movies keyed by user id. Each movie in this dataset is associated with one or more categories chosen from a dictionary of 20 movie genres. These genres include categories like comedy and thriller as well as a “no genres listed” category.

	MSD	**MovieLens 25M**
Number of users	648,986	162,541
Number of distinct categories	100	20
Average entries per user	2.5	153
Median entries per user	2	70
Average number of category labels per song/movie	5.01	1.79

Table 1. Description of the public datasets

Feature extraction

In order to map users into a vector space, we consider a two stage process:

- Song/movie feature extraction: For the MSD dataset we encode each (song, user) pair as a feature vector v where the value of element v_i is the weight of category i for this song, multiplied by the number of times this song was listened to by the user. The extraction of movie features is similar, except we multiply the category weight by the rating the user gave this movie instead of the number of times the song was listened to.
- Feature vector aggregation: To aggregate all feature vectors associated with a user we simply take the average of the vectors.
- Centering. Finally, we center all feature vectors to ensure the dataset has mean zero.

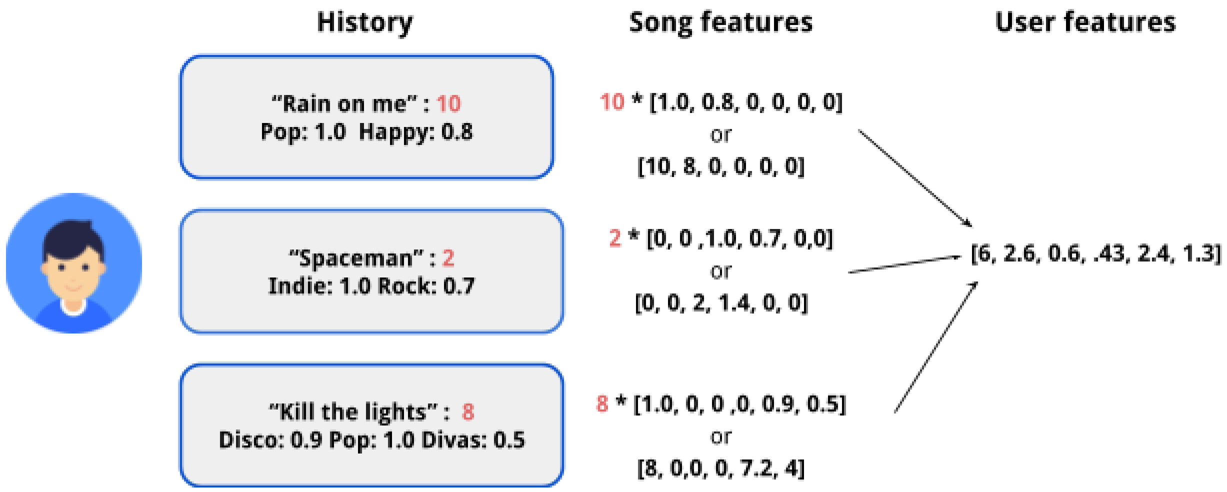


Fig 3. Description of the feature generation process for the MSD dataset

Evaluation

We are interested in measuring the utility of cohorts for varying levels of anonymity. To measure the quality of a cluster we use the average cosine similarity (or normalized dot product) between all users in the cluster and the centroid of the cluster. This metric is always between -1 and 1. The cosine similarity of two vectors is -1 if they are diametrically opposed, 0 if they are orthogonal and 1 if they are identical. To measure privacy we look at the 2% quantile of the cohort size distribution weighted by the number of users in the cohort. This metric corresponds to the level of k-anonymity protection provided to 98% of users. Ideally, we want a clustering algorithm that generates large cohorts with cosine similarity close to 1. As a baseline we compare to a clustering algorithm that randomly assigns users to cohorts. The results of this experiment can be found in the figure below.

The plots show that a fully centralized clustering approach, non-surprisingly, outperforms the other two methods and can achieve very high intra-cluster similarity. Nevertheless, it is promising to see that a fully decentralized approach can generate cohorts achieving approximately 85% of the quality of a fully centralized algorithm.

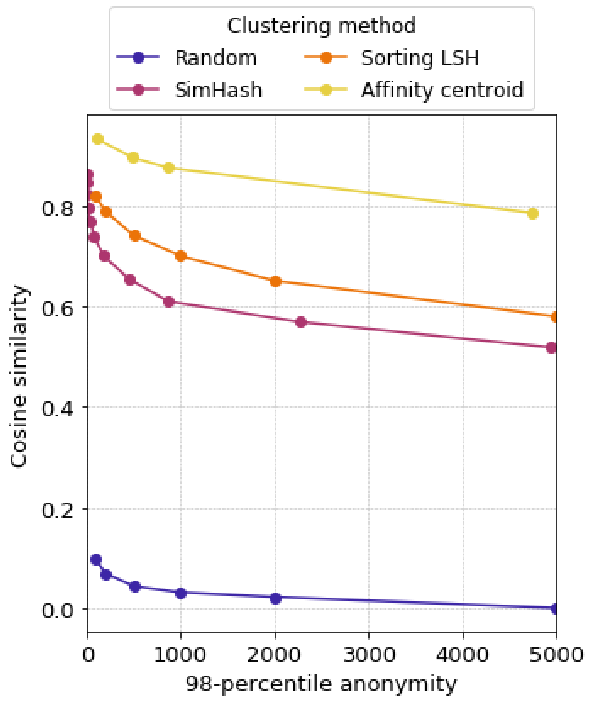


Fig 4a. Results for the MSD dataset

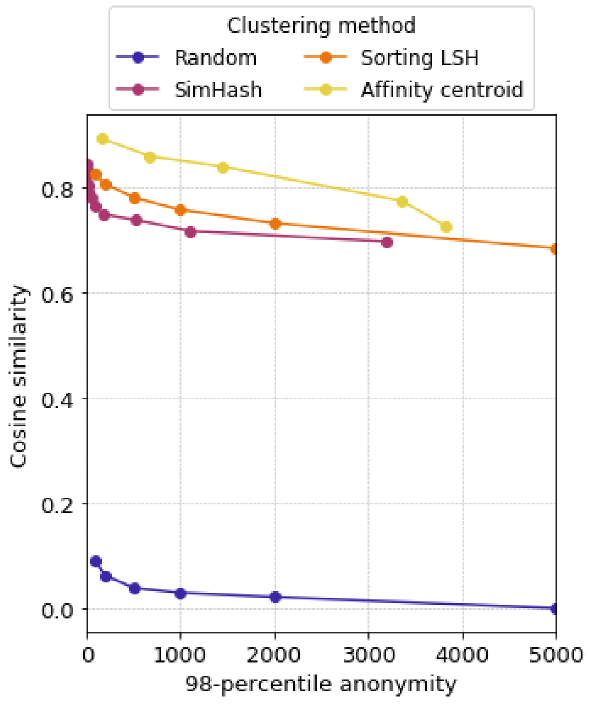


Fig 4b. Results for the MovieLens dataset

Visualization

The previous plots provide us with a quantitative way of comparing cohorts, in this section we attempt to visualize the cohorts obtained by the different algorithms to understand the semantic meaning, if any, of these cohorts. In practice, this would correspond to building interest profiles based on cohort ids instead of third party cookies.

To visualize cohorts we look at the average of all users in a cluster, or its centroid, and generate a word cloud of the categories in this cluster, where the font size corresponds to the weight of the category in the centroid vector.

MSD

We begin by displaying the cohorts generated by randomly grouping together users. It is immediately obvious that this random clustering is useless for classifying users since there are multiple highly weighted and unrelated music categories.

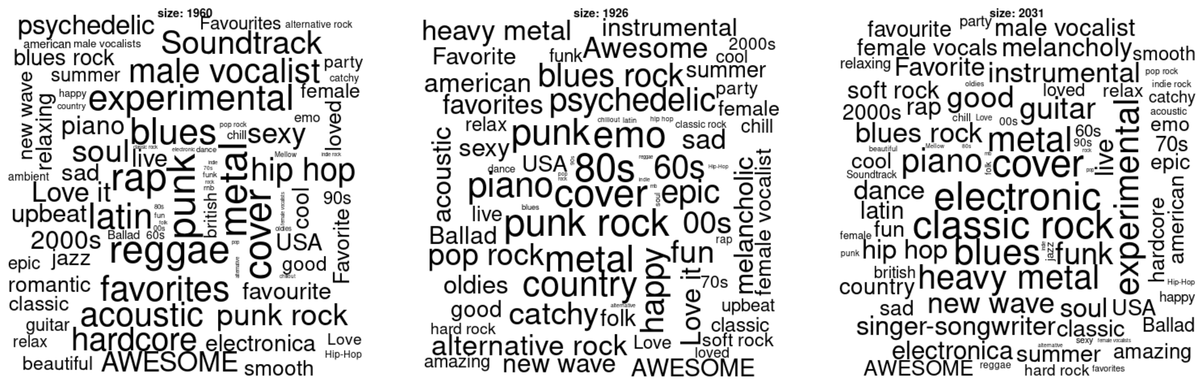


Fig 5. Word clouds for random cohorts

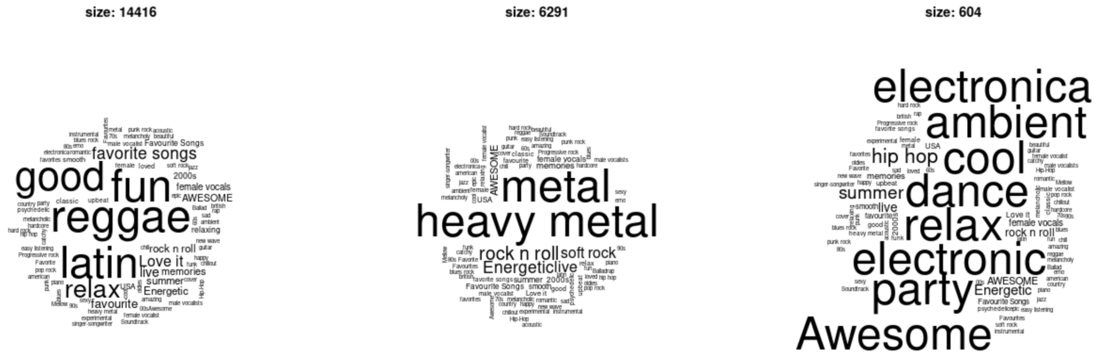


Fig 6. Word clouds for SimHash cohorts using 8 bits

When using SimHash, we can see that some cohorts are very well defined, such as the “metal” cohort. This is achieved even though the cohort size is larger than the cohorts generated randomly. The first and third cohorts do not represent a single music style but rather two somewhat related styles. The first one seems to represent reggae and latin music while one could infer the third cohort contains mostly users who listen to house and dance music.

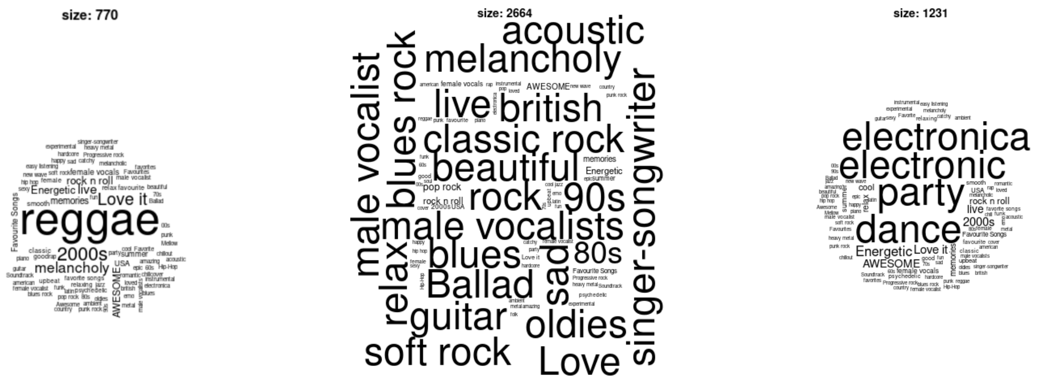


Fig 7. Word clouds for affinity clustering

Finally we show the cohorts generated by using affinity clustering. Here the cohorts are smaller than the ones generated by SimHash and also better defined. Unlike SimHash cohorts who had two leading genres, the first and third cohorts generated by affinity clustering seem to have a single genre. The second cohort seems to have multiple categories associated with it, yet closer inspection seems to suggest that this cohort consists of users who listen to artists who play soft, mellow rock ballads.

MovieLens

We now turn our attention to the MovieLens dataset. As before, we begin by showing cohorts formed by randomly grouping users together. It is clear that such cohorts provide us with no semantic meaning: all cohorts look roughly the same and no particular genre dominates. We can compare these with the cohorts generated using SimHash

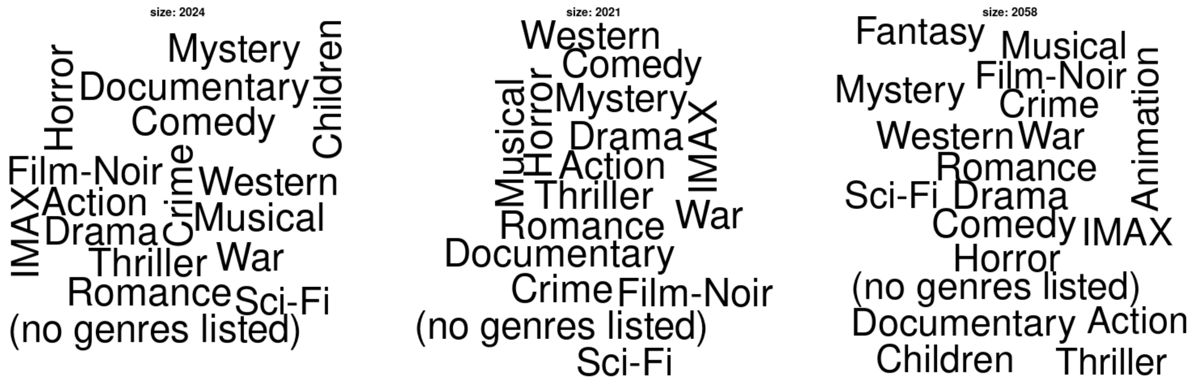


Fig 8. Word clouds for random cohorts

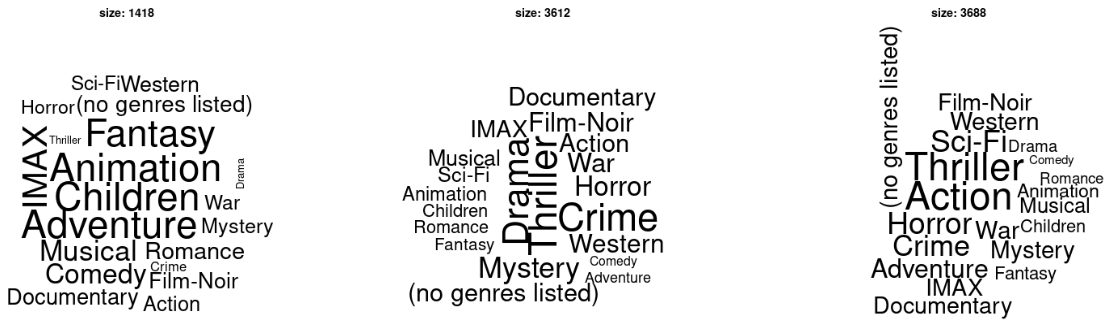


Fig 9. Word clouds for SimHash with 8 bits

The first thing to notice is that unlike the random cohorts, these cohorts have few leading genres. Moreover, they seem to complement each other. For instance, the first cohort main topics are fantasy, animation, children and adventure. All these genres clearly relate to children movies such as “Toy Story”.

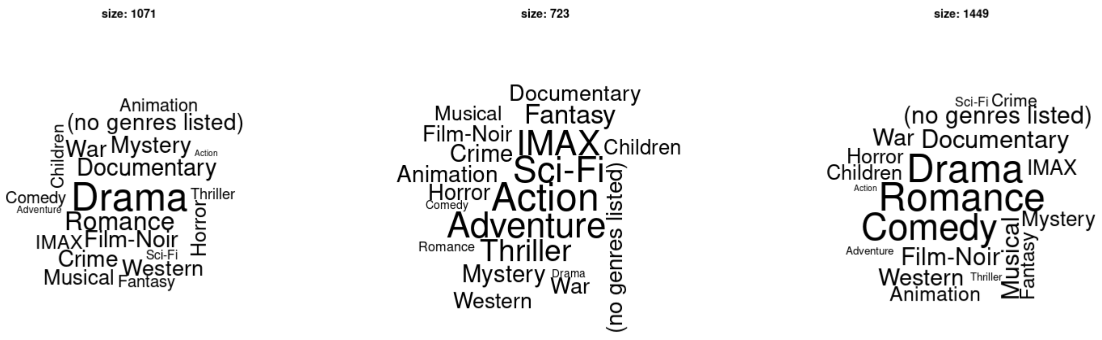


Fig 10. Word clouds for affinity hierarchical clustering

Finally, we show the cohorts generated by using affinity clustering. While they seem to be slightly better defined than SimHash, the differences are not as stark as in the MSD dataset.

The results shown in this section demonstrate the viability of having a fully decentralized clustering algorithm based on a very simple algorithmic technique. However they also show, rather unsurprisingly, centralized clustering algorithms do outperform SimHash.

Google Display Ads Dataset

Having established the viability of clustering algorithms on public data we turn to evaluating their viability for ads personalization. For this purpose we conducted extensive evaluations on a proprietary dataset.

Dataset description
The dataset consists of a de-identified collection of URLs from publishers in the Google Display Network collected across 7 days. We emphasize that there is not a one-to-one correspondence between user and pseudonymous ids as a user can be associated with multiple ids. For instance, one per device used.

Feature extraction

Unlike the public datasets we previously evaluated, features can be extracted in multiple ways from this URL-based dataset. Here we discuss only a few possibilities.

Domain One-hot encoding
Under this feature extractor, each URL visit is encoded by its domain only. The feature vector representing a user is simply a one-hot encoding of all the domains they visited. That is, it is a sparse vector where all domains visited by a user get assigned a 1, regardless of the number of times the domain was visited.

Domain TF-IDF encoding
Here again we encode each URL visit by its domain. However, instead of weighting all websites in the same way we use TF-IDF scoring. That is, if a user visits a domain frequently it gets a higher score but if the domain is very popular - multiple users visit the same domain - it is assigned a lower weight. The feature vector representing a user is simply the sparse vector of scores associated with each domain visited by the user.

Topic categories
A better way of generating feature vectors is by categorizing the websites into topics. For instance expedia.com is a travel website. Categorization is done using a tool similar to the topic categorization API of Google Cloud[4]. This categorization corresponds to a hierarchy of 3 levels, where each level describes a more specific category. Here is one example of a hierarchy:

/Arts & Entertainment
/Arts & Entertainment/Performing Arts
/Arts & Entertainment/Performing Arts/Acting & Theater

The categorization API assigns a website to at most 5 categories and provides weight for each category proportional to how well the category describes the website. The feature vector

[4]Google cloud URL categorization: https://cloud.google.com/natural-language/docs/categories

representing a user is a sparse vector where each entry in the vector corresponds to the average weight of a topic category across all URLs visited by a user.

Clustering Technique

We use the SimHash algorithm using the three different features described above:

1. domain one-hot encoding
2. domain tf-idf encoding
3. topic categories (we stop at vertical depth 3).

Evaluation

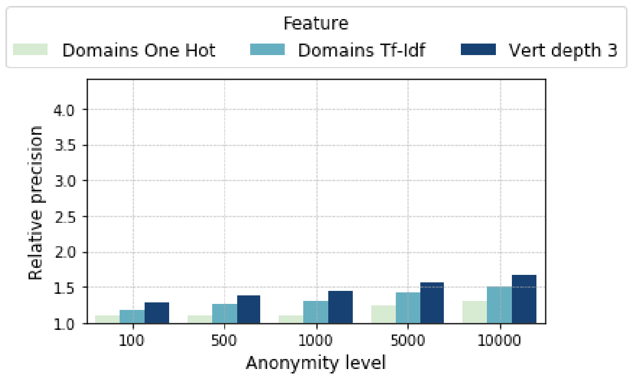
We measure the viability of using cohorts as a targeting tool as follows:

1. **Building interest profiles.** For each cohort we categorize all websites that would be tagged by the same cohort id and aggregate all topic category weights. We then define the cohort interest profile as the top 10 categories.
2. **Building conversion profiles.** We use 7 days worth of conversion data to generate a conversion profile for each user. This conversion profile consists of the set of website topics where the user converted over a period of 7 days.
3. **Evaluating predictive power.** We use the user interest profile to predict whether a user will convert on that topic (from raw conversion data). The prediction is correct if a conversion actually happened in the next 7 days. We measure precision and recall for these predictions.

To measure performance, we drop all profiles that do not meet a k-anonymity threshold and replace them with a cluster profile that corresponds to the most popular verticals:

- /Apparel & Accessories/Women's Apparel
- /Employment
- /Financial Services/Investment Services
- /Autos & Vehicles/Motor Vehicles/Motor Vehicles (Used)
- /Dating Services
- /Telecom/Mobile Phone Service Providers
- /Real Estate/Residential Properties/Residential Properties (For Sale)
- /Real Estate/Residential Properties
- /Consumer Electronics/Mobile Phones
- /Autos & Vehicles/Motor Vehicles

To compute a baseline, for each desired anonymity level we generate cohorts by randomly clustering users together in order to clear the anonymity threshold.The plot below shows relative precision/recall compared to the random baseline.



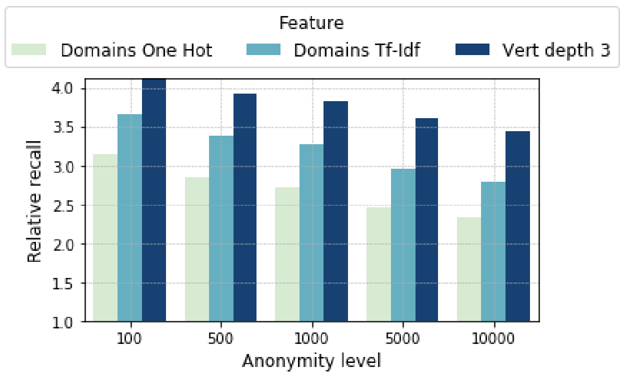
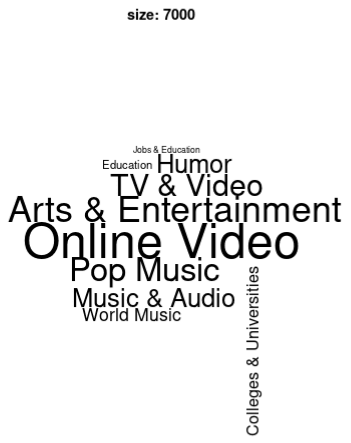


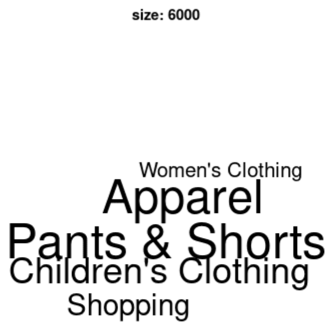
Fig 11a. Relative precision of different features on Simhash Clustering compared to a random baseline.

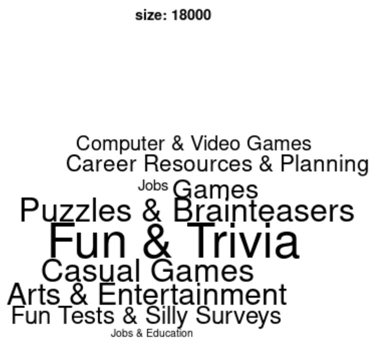
Fig 11b. Relative recall of different features using Simhash Clustering compared to a random baseline.

What we can see from these plots is that even at very high anonymity levels, grouping users together into cohorts with similar interests yields up to 3.5 times better recall (for k=5,000 using Vertical depth 3 classifier) than randomly clustering users together. We also see that using topic categories as a feature yields the best performance. Something that might be counterintuitive is to see precision increasing as the anonymity level increases. This is expected since increasing anonymity constraints leads to replacing more cohort profiles by the default profile. This implies that there is a smaller number of topics to predict or fewer ways in which to be “wrong”. While we did not report the results of other clustering techniques (SortingLSH and Affinity Clustering) on this dataset, we found the quality ranking of these algorithms to be similar to the MovieLens and Million Song Dataset.

To conclude the analysis of the performance of this algorithm on our proprietary data, we present some qualitative results for the topic categories feature generation. Below we present word clouds representing the average profile associated with a cohort. These clouds show there is a tangible semantic representation to the cohorts we generated, even though the number of users in each cohort is large. Indeed, we can easily infer that the first cohort is composed of users who are most interested in entertainment genres. The second one is a cohort of users who commonly browse apparel websites and the last cohort seems to be related to games and trivia. In practice, the true value of cohort assignment is amplified when it is crossed with the context of the current page.







Conclusion

In this paper, we proposed a series of cohort id assignment algorithms that use multiple clustering techniques (distributed and centralized) and evaluated on different datasets (freely available and proprietary) using various ways of choosing and computing features. We also proposed a technique for evaluating utility against various k-anonymity measures. For the task of predicting conversions, FLoCs was shown as a very informative signal, and we can achieve significant improvements in recall and precision over randomly assigning users to cohorts even at high anonymity levels.

Based on our results, we believe that the SimHash technique on topics could be a good starting point for experimenting within the browser. We also encourage other ad-tech providers to evaluate some of our proposed algorithms on their proprietary datasets. We look forward to hearing new ideas from the broader community on how this system can be further improved.