Sparse User Personalization via Lookalike Clustering: A Cold-Start Solution

Shashank S, contact@knhash.in
Ritika Kumari, ritikakumari1302@gmail.com

***Abstract*—'Cold start' is a potential problem in recommendation systems where the system does not have enough information about the user or content to make optimal recommendations. This information typically takes the form of user interactions with the system. In this paper we present an approach to recommendation that is built on low cardinal input user-feature space. The idea is to emulate (and hence build a lookalike model) of "dense" users with the hypothesis that "users of the same feather flock together". We justify the thought process behind the approach with some preliminary analysis. We then specify the end-to-end system along with the underlying model specifications - which is based on K-Means Clustering [Steinhaus et al.(1956)] [Lloyd(1982)]. The real-time recommendation system service itself is lightweight and operates with low latency overhead. Finally, we showcase the experimental results on a real-world dataset of 400k users and compare the performance with existing recommendation models at Glance.**

***Index Terms*—exploration, cold-start, recommendation system, lookalike, real-time, clustering, k-means**

I. Introduction

Glance(https://glance.com) is a content delivery platform, serving more than 200 million daily active users across Android lock screens. Content pieces can be of both the types, long-living (evergreen) and short-lived (ephemeral). With the primary interface eschewing discoverability in favor of a more organic experience, personalization plays a critical role.

A key component driving personalization systems is user interactions. Cold users are users who have just entered our system. Sparse users have had some, sparse, interactions with our system. For both these kinds of users, interactions cannot be depended upon for personalizing the content. It is just not a rich enough source of data yet. Thus, the personalization problem can be split into two parts, to be tackled differently:

- when we have enough user interactions to drive our models, we call them Dense users,
- and when we do not have enough user interactions, we call them Sparse users

Sparse users are important, after all every user was once a sparse user. For them, the goal of a personalization system is twofold – get them engaged enough so a dense user model can take over and "explore" enough content identify the user's interests. Exploration here means to serve a variety of content to the users to gauge their preference. Once the interests are established, we can "exploit" this preference to serve content to the user, which they are more likely to enjoy and engage with.

In this paper we present a methodology to recommend content to sparse users. We will start by defining the problem statement and the unique constraints associated with it. We will walk through the initial analysis that led to the hypothesis, the implementation specification, the experimental setup and our empirical results. The implementation details will focus on shaping a solution from our preliminary analysis and building a low latency prediction service. Finally, we will revisit the problem statement to see where we go from here.

II. Background

The content pieces we recommend at Glance are called Glances, grouped together in contextual bundles called Bubbles. They cover a lot of categories – sports, fashion, entertainment, etc. – and a variety of languages. It could be in the form of a video, article or just a tweet. Each Bubble is associated with a category, as defined by the editor and moderators who publish the content.

Cold users are users who have just entered our system. Sparse users have had some, sparse, interactions with our system. For both these kinds of users, interactions cannot be depended upon for personalizing the content. It is just not a rich enough source of data. Users stay on a system because the content is constantly engaging. At Glance, a separate analysis showed that users for whom we serve content closest to the categories they like, interact the most. So, the cold-start problem for us can be simplified to identifying the category affinity of the user and leveraging it in personalization.

But we do not have the interactions of these users, not yet.

What we do have is a low cardinality user feature space: user demographics like gender and tier of city, device specifications like price and screen resolution and one rich meta feature: category preference information – details about the kind of content a user wants to see – on their device. This category preference is collected as part of the onboarding process.

Our solution approach is to build a lookalike model based on the following idea: there are clusters of users who exhibit similar interaction behaviors. With these clusters, we can leverage their group interactions to power our personalization system. These clusters will be identified among the Dense users, using the five user features mentioned above. We then tag a cold/sparse user to a cluster number and use the interaction characteristics of the cluster to rank the Bubbles for the user.

Shashank and Ritika were part of the Machine Learning Engineering team, Feed Personalization, of Glance Digital Experience Private Limited (InMobi), India.

Thanks to Titas, Suba and Arkid, for guiding us out of tough spots.
Manuscript received October 19, 2022; revised June 1, 2024.

III. Related Work

Most of the work in recommendation systems has been around collaborative filtering-based approaches [Zhang et al.(2014)]. But in practice, the effectiveness of collaborative filtering models is limited by the sparsity of interaction matrix of historical users and cold start users [Ahn(2008)]. The sparsity is majorly because users show explicit interest in only a few pieces of content. With the increase in time duration, the sparsity increases even more. Many have attempted to overcome this limitation by adopting different approaches such as clustering or dimensionality reduction, whose key motive was to use low-dimensional matrix instead of original high dimensional matrix. Instances of these kind of approaches can be seen in [Cheng and Church(2000)] [Barragáns-Martínez et al.(2010)] [Luo et al.(2014)] which primarily used bicluster algorithms, singular value decomposition and non-negative matrix factorization.

Working on the same lines, authors in [Jin and Han(2020)] combined the latent-factor approach and the clustering approach to obtain a finer representation of their interaction data. Latent factorization helped to break the user-item matrix into user preference matrix and item preference matrix, on which clustering is done individually. A collaborative filtering model is then plugged in for predictions.

In [Das et al.(2014)], the authors adopted a slightly different approach, wherein, a DBSCAN clustering algorithm is employed to cluster users. Users are served items based on different voting algorithms for each cluster separately. The users in specific clusters are recommended items based on the ratings of the users of that cluster only.

A similar work is proposed by the authors in [Brodinová et al.(2019)], where they used K-Means clustering along with a weighing function which assigned weights to each of the observations. They further used the lasso-type penalty in their objective function to reduce the dimensionality of their data.

IV. Analysis

The hypothesis we are working with is that users having similar characteristics behave/interact similarly. Essentially, there are clusters of users that we can identify based on our input features. For the sake of this experiment, we decided to go with a simple clustering model, K-Means, using Euclidean distance as our distance metric. Our training dataset consisted of the control set of users which made up 10% of our user base. We filtered for dense users within this to arrive at our final dataset. We concentrated on the dense users' data as we were trying to replicate the dense user behavior onto similar sparse users.

A. Feature Engineering

As a first step, we analyzed user's category preference and demographics data. Under demographic data, we had 4 features to work with:

- `gender of the user`: a binary variable, identifying the predicted gender of the user
- `tier of the city`: a categorical variable having three values
- `price`: a continuous variable, price of the Android phone
- `dpi`: a continuous variable, screen resolution of the Android phone

Gender of the user was a derived feature based on various other device characteristics. For the continuous features such as price and dpi, we first bucketed them into defined ranges and encoded them as one-hot vectors. The tier of city and gender were similarly one-hot encoded. The user's category preference comes as an 18-dimensional vector which we encoded as a multi-hot vector and normalized it.

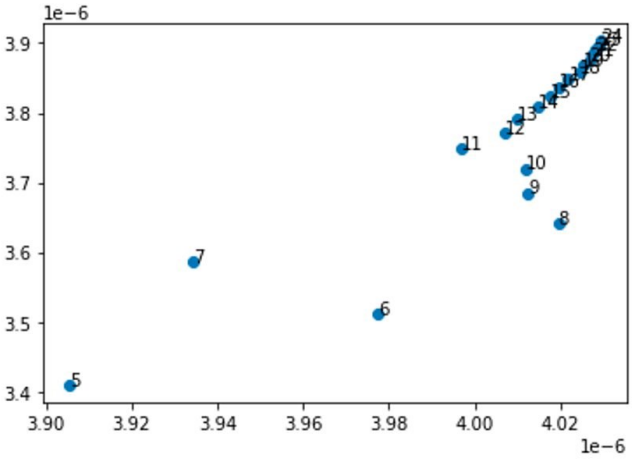

Fig. 1: Inter-variance (x-axis) vs intra-variance (y-axis) for SVD. The optimal number of dimensions comes out to be 6.

B. Dimensionality Reduction

On concatenating the normalized multi-hot category vector with the rest of the one-hot encoded features, our feature space was too large and sparse. Clustering the users via k-means clustering is unlikely to give meaningful clusters with such sparse and high dimensional data [Nur'Aini et al.(2015)]. Here, Singular Value Decomposition (SVD) without mean normalization turned out to be the most suitable approach. To decide the optimal number of reduced dimensions, we plotted the inter and intra variance of the SVD matrix for a defined range of dimensions. The optimal dimension should have high inter variance (x-axis) and low intra variance (y-axis) scores, which came out to be 6 for us as shown in Figure 1.

C. Finding optimal number of clusters

We plotted an elbow graph for Within-Cluster Sum of Square (WCSS) [Bagirov(2008)] vs number of clusters to determine the optimal number of clusters for our clustering. WCSS is the sum of the squared distances between each point in a cluster and its centroid. As the number of cluster increases, the WCSS value decreases, giving us an elbow point where the WCSS value declines the most.

The normalization of preference categories at a global level was not helping the model generalize, it was not calibrating

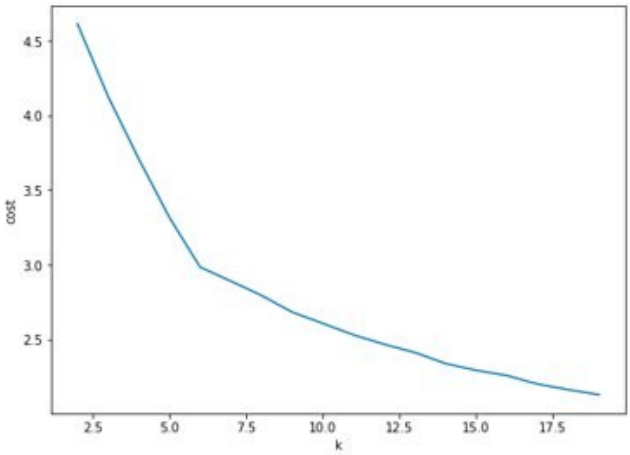

Fig. 2: Plot of the elbow curve, WCSS vs number of clusters. The optimal number of clusters comes out to be 6.

the model for different user groups. We therefore incorporated the price of the device into the preference category vector, by normalizing the multi-hot category vector for each price bucket. Our reasoning here was that people buying devices in certain ranges would tend to have similar tastes. So, the preference categories vector now indicated how different the preferred categories were for a particular user as compared to other users in the same price bucket of the device.

This helped us greatly in tuning the model. After all the iterations, the optimal number of clusters came out to be 6, as evidenced by the faint elbow in Figure 2.

On checking the feature distributions of the clusters, we were able to see clear distinctions, which meant that we had meaningful clusters. The distinction was clear enough to engineer user personas out of it, reproduced in Table I.

D. Correlating with interaction data

Our immediate next step was to check how well these clusters relate to the user's interactions. More precisely, we wanted the users of these clusters to have different behaviors. For our dense users clustered in our training dataset, we looked at their interaction scores over a 30-day period. These interactions were calculated over a binary set of rewards, given when a user positively interacts with Bubbles. The scores were aggregated for the categories of the Bubbles interacted by the user, and then aggregated again at a cluster level. We thus prepared a weight matrix, which is the mapping between cluster numbers and mean interaction score of categories, for the users in those clusters.

Figure 3 represents this mean weight matrix. Here, the vertical bands mean that all the clusters behave generally similarly across the interaction categories. The very slight horizontal banding means there is a variation in the interaction behavior. But that was too slight a variation to have any predictive power.

Since mean interaction scores were possibly skewed due to outliers, we repeated the same exercise with median interaction scores and now the horizontal banding became more distinct, as shown in Figure 4. Notice how cluster 1 and 4 stand out generally in all categories, 0 and 5 in automotive, and cluster 1 in health and fitness. This highlighted a correlation between the clusters and the interaction categories that could be leveraged for sparse users' recommendation.

V. Implementation

A. The end-to-end solution

The lookalike model works on the hypothesis that sparse users belonging to the same cluster as predicted by our k-means model (trained on dense users) should exhibit similar interaction preferences. By taking this preference into consideration in our ranking, we should notice a better reward rate and higher engagement from our sparse users – eventually turning them into dense users.

Our recommendation system would therefore consist of:

- Identifying the cluster number that a cold or sparse user belongs to
- Fetching the associated weight vector for the categories of the content
- Ranking the Bubbles by using this vector as the defining rubric

For the sake of our experiment, we will test this A/B on a set of 400,000 active users. The architecture overview is provided in figure 5.

B. Specification of the prediction service

Our recommendation systems at Glance operate under some tight constraints:

- Latency of the prediction model should be less than 150ms
- The number of Bubbles ranked per user request can go up to 400 Bubbles per request
- At any given time, we have roughly 4000 content pieces in our repository to be served, as they keep getting refreshed.

We have one pipeline to train our clustering model on our dense users. We do not expect a lot of drift in the data, so this model could be refreshed as lazily as every fortnight. The users belong to our control subset, so that other model biases are not introduced into the dataset. This subset forms 10% of our active user base, which has a very simple serving algorithm which we will compare ourselves with, called *Pacing*.

A second pipeline runs at a daily cadence, to identify the cluster number of the users in our experiment. This pipeline loads the k-means model and runs clustering on target set of users. It then uses the weight matrix to identify the corresponding category weights for the user. The `[user_id, category_weight]` mapping is saved in a low latency cache. This pipeline takes about 7 minutes to run.

Our prediction service takes three inputs:

- `user_id`: a unique identifier for the user in our system
- `bubble_count`: number of Bubbles to be ranked
- and `live_bubbles`: a candidate list of Bubbles to rank

It fetches the `category_weight` of the user from the low latency cache. It then looks up two attributes for every Bubble in `live_bubbles`: category and global popularity

TABLE I: User personas emerging out of the clusters

Cluster	DPI	Tier	Gender	Price	Preference Categories (ordered by priority)
0	High	Mix	Male	Mix	Finance, Productivity, Entertainment
1	High	T1 & T3	Dominant Male	Mix	Shopping, Finance, Business, Social
2	High	Mix	Dominant Male	Middle	Finance, Social
3	Very Low	Mix	Mix	Low	Finance, Social, Entertainment
4	Low	Mix	Mix	Middle	Social, Entertainment, Finance, Shopping
5	Very High	Mix	Mix	High	Shopping, Finance, Entertainment, Productivity

int_cat	#animals	#automotive	#business	#comedy	#cricket	#entertainment	#fashion	#food	#fun_facts	#games	#health_n_fitness	#international	#music	#national	#sports	#talent	#technology	#travel	No_Category
cluster_num																			
0	0.376841	0.276393	0.260318	0.281347	0.178658	0.438879	0.55628	0.397835	0.219424	0.380594	0.330383	0.334385	0.484921	0.353613	0.298105	0.55687	0.254009	0.367256	0.308276
1	0.435383	0.306107	0.316293	0.275663	0.19964	0.512705	0.609592	0.454114	0.210348	0.510341	0.405008	0.4054	0.505235	0.380032	0.320662	0.605215	0.28244	0.438502	0.348912
2	0.381088	0.273385	0.264718	0.266602	0.158944	0.448423	0.551591	0.397292	0.196712	0.42617	0.327957	0.340424	0.494015	0.352842	0.297849	0.555527	0.237375	0.382692	0.314876
3	0.388113	0.267751	0.263517	0.257788	0.167391	0.454342	0.555929	0.403586	0.202433	0.443101	0.321806	0.338456	0.486554	0.347019	0.294682	0.579498	0.248838	0.390251	0.317822
4	0.414838	0.295563	0.279633	0.287614	0.177581	0.481158	0.567132	0.44457	0.210853	0.473407	0.353183	0.37231	0.503655	0.36552	0.300378	0.589291	0.261158	0.414864	0.328566
5	0.395873	0.281163	0.272561	0.24167	0.181101	0.453154	0.559639	0.412251	0.220078	0.39383	0.355596	0.356928	0.491336	0.359657	0.303516	0.582008	0.262579	0.395577	0.316386

Fig. 3: A map of cluster number and the mean interaction score of the users belonging to the cluster

int_cat	#animals	#automotive	#business	#comedy	#cricket	#entertainment	#fashion	#food	#fun_facts	#games	#health_n_fitness	#international	#music	#national	#sports	#talent	#technology	#travel	No_Category
cluster_num																			
0	0.29	0.08	0	0	0	0.43	0.6	0.33	0	0.29	0	0.17	0.5	0.33	0.25	0.63	0	0.25	0.26
1	0.4	0	0	0	0	0.5	0.8	0.5	0	0.5	0.14	0.25	0.5	0.33	0.28	1	0	0.33	0.33
2	0.3	0	0	0	0	0.44	0.6	0.33	0	0.33	0	0.14	0.5	0.33	0.25	0.6	0	0.25	0.27
3	0.33	0	0	0	0	0.46	0.6	0.33	0	0.4	0	0.13	0.5	0.33	0.25	0.75	0	0.29	0.29
4	0.33	0	0	0	0	0.5	0.63	0.46	0	0.5	0	0.19	0.5	0.33	0.25	0.75	0	0.33	0.3
5	0.33	0.08	0	0	0	0.44	0.64	0.33	0	0.33	0	0.2	0.5	0.33	0.25	0.67	0	0.33	0.28

Fig. 4: A map of cluster number and the median interaction score of the users belonging to the cluster

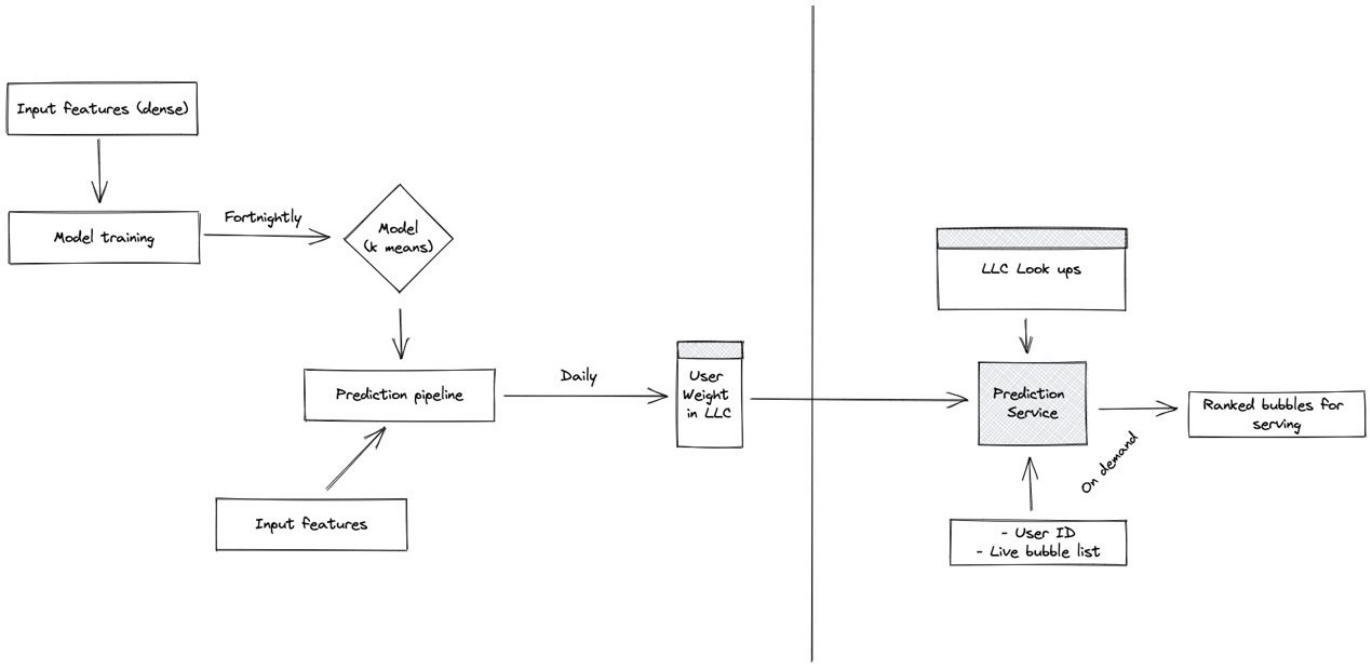

Fig. 5: Overview of the end-to-end system design

of the Bubble. The global popularity of a Bubble is sampled from a beta distribution composed of the positive and negative interactions of the users on that Bubble, over a period of the previous 30 days. All the lookups are locally cached to avoid repeated network calls.

The `category_weight` of a user identifies a category affinity. We need to translate this to a Bubble level score. To do that we envision `bubble_count` number of slots to fill with `live_bubbles`. The `category_weight` tells us which category the user is more likely to interact with. Multinomial sampling with the different categories weighed differently lets us divide bubble count into category slots. A category slot is filled in with the most popular bubble in the live bubbles list, according to global popularity score.

The category weight only tells how likely a cluster is to interact with the categories and should not be interpreted as order of interaction. Say "fashion" had 0.5 and "sports" had 0.3 as their category weights for a certain cluster. This doesn't mean fashion content should come "first" while ranking the content, all this means is that fashion was preferred more as a category. Therefore, we do not encode the category weight into the final score associated with the Bubble and use it only to dictate the number of Bubbles. The popularity score serves as the final step to attach a score to every bubble.

Two further tweaks make this ready for production:

- A 20% reservation to explore categories uniformly, to better identify category affinity
  - Essentially our model operates on 80% of `bubble_count`, with 20% of the `bubble_count` sampled from a uniform distribution of category weights
- A fallback mechanism for when `category_weight` is unavailable for a user
  - Because the pipeline runs at a daily cadence there would be users who would not be touched by the pipeline yet, and hence have no `category_weight`. For such users we fallback to scoring the bubbles basis their global popularity, sampling from the global interaction distribution.

Because the final prediction service does only light weight calculations it is extremely fast. It can maintain a steady state p99 latency of 60ms.

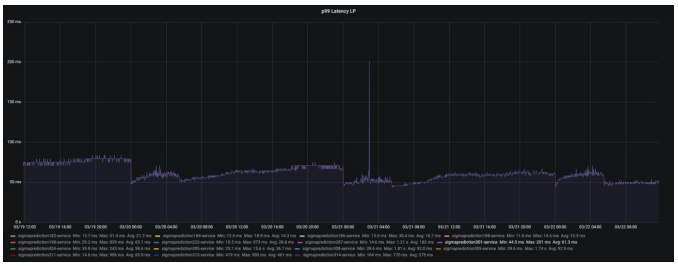

Fig. 6: Grafana dashboard showing the p99 latency of the prediction service, at a steady state of around 60ms

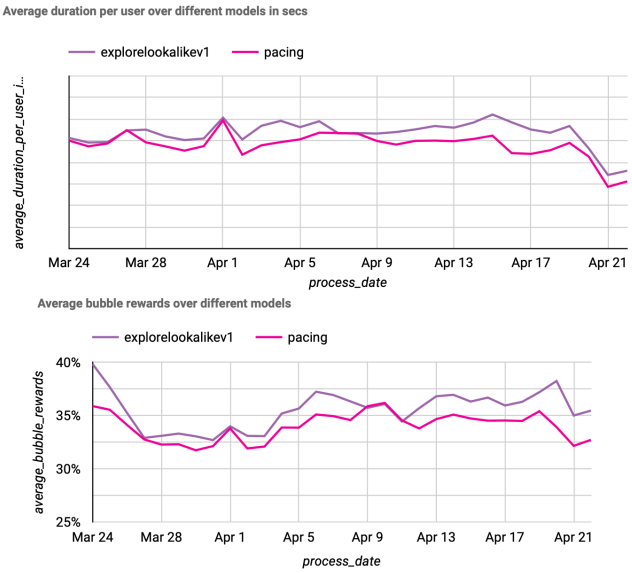

Fig. 7: Time spent (top) and Reward Rate (bottom) of our model as compared to Pacing, over all cohorts

VI. Results

Our primary model of comparison is called Pacing. It is a simple rule-based model that "paces" or limits the Bubbles served to the user based on certain rules, and acts as the default baseline to compare performance against at Glance. We also benchmark our model, called `explorelookalikev1`, against some of the other models active during that period. These include a user level Logistic Regression model (`userlrmodel1dot1`) and a basic version of a Multi Armed Bandits model (`basemodelhighlights`). We will look at the performance both at an overall level and at a cohort level; specifically at cold users.

To measure engagement, we have two metrics:

- `Time Spent`: Average duration a user spends on Glance in seconds, per day
- `Reward Rate`: Average reward rate of the model, per day. Either at a Bubble or Glance level.

For the sake of confidentiality, the actual values of Time Spent have been masked. Reward Rate is a score calculated over an aggregation of positive interactions. These include such aspects as dwell time, explicit positive signals and virality.

`explorelookalikev1` consistently performs better than pacing across both the metrics, as seen in Figure 7. On average it gets a 11.68% delta over Pacing. The Reward Rate of our model is higher by 4.5% on average.

When compared to the other models (Figure 8) during the same period, our model routinely beats them on both the metrics, although by a smaller margin.

Note that this across all the cohorts – cold, sparse, and dense. Because our model was trained on Dense users it was bound to perform well for them. It will not be sustainable, however, as at this stage it is only identifying popular content

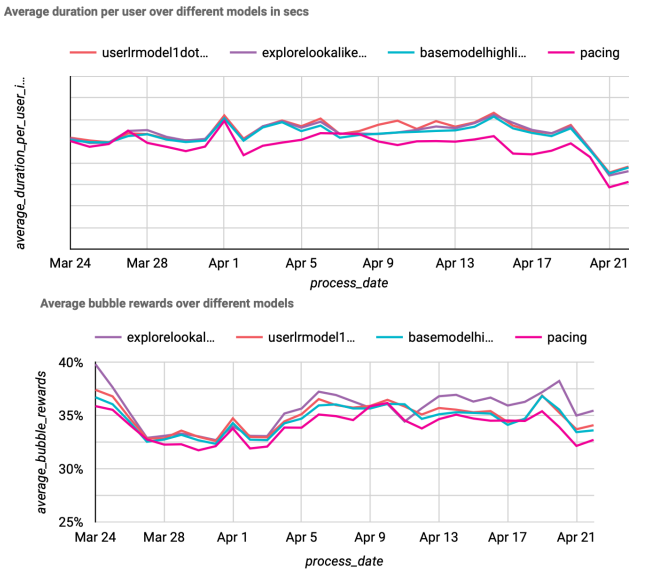

Fig. 8: Time spent (top) and Reward Rate (bottom) of our model as compared to other models, over all cohorts

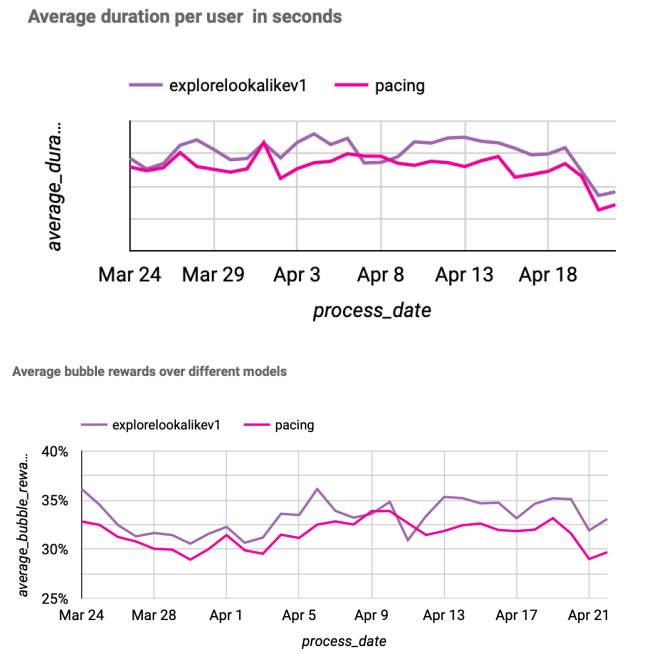

Fig. 9: Time spent (top) and Reward Rate (bottom) of our model as compared to Pacing, for Cold Users

in specific categories for the users. If scaled it will not be able to explore content as well as a standard MAB.

For Cold users (Figure 9), the model performance comes out more distinctly. An average lift of 18.38% in Time Spent and 5.76% in Reward Rate as compared to Pacing. We also see competitive (and, at worst, at par) performance with respect to the other models (Figure 10).

The improvement is especially stark when looked at the

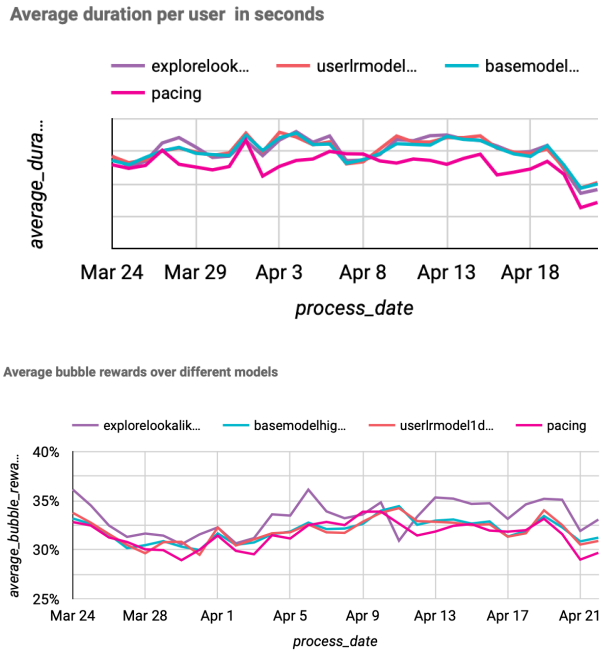

Fig. 10: Time spent (top) and Reward Rate (bottom) of our model as compared to other models, for Cold Users

Reward Rate at Glance level, for Cold users (Figure **??**. An average lift of 51.8% as compared to Pacing and consistently beating out the other models. Our model, thus, performs well across all the metrics, especially so on Cold users.

VII. Conclusion and Future Work

In this paper we showcased our clustering-based lookalike approach for recommending content to Cold and Sparse users. We show empirical evidence that the approach works, based on a real-world A/B experiment over a period of 30 days.

For future iterations, the system is being modified to incorporate new, richer features to better define the clusters. Experiments are also underway to test other clustering methods like Spectral Clustering, which should bring out the latent representation of the clusters better.

REFERENCES

[Ahn(2008)] Hyung Jun Ahn. 2008. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information sciences* 178, 1 (2008), 37–51.

[Bagirov(2008)] Adil M Bagirov. 2008. Modified global k-means algorithm for minimum sum-of-squares clustering problems. *Pattern Recognition* 41, 10 (2008), 3192–3199.

[Barragáns-Martínez et al.(2010)] Ana Belén Barragáns-Martínez, Enrique Costa-Montenegro, Juan C Burguillo, Marta Rey-López, Fernando A Mikic-Fonte, and Ana Peleteiro. 2010. A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition. *Information Sciences* 180, 22 (2010), 4290–4311.

[Brodinová et al.(2019)] Šárka Brodinová, Peter Filzmoser, Thomas Ortner, Christian Breiteneder, and Maia Rohm. 2019. Robust and sparse k-means clustering for high-dimensional data. *Advances in Data Analysis and Classification* 13, 4 (2019), 905–932.

[Cheng and Church(2000)] Yizong Cheng and George M Church. 2000. Biclustering of expression data. In intelligent systems for molecular biology.

[Das et al.(2014)] Joydeep Das, Partha Mukherjee, Subhashis Majumder, and Prosenjit Gupta. 2014. Clustering-based recommender system using principles of voting theory. In *2014 International conference on contemporary computing and informatics (IC3I)*. IEEE, 230–235.

[Jin and Han(2020)] Yingjie Jin and Chunyan Han. 2020. A music recommendation algorithm based on clustering and latent factor model. In *MATEC Web of Conferences*, Vol. 309. EDP Sciences, 03009.

[Lloyd(1982)] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[Luo et al.(2014)] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.

[Nur'Aini et al.(2015)] Khumaisa Nur'Aini, Ibtisami Najahaty, Lina Hidayati, Hendri Murfi, and Siti Nurrohmah. 2015. Combination of singular value decomposition and K-means clustering methods for topic detection on Twitter. In *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 123–128.

[Steinhaus et al.(1956)] Hugo Steinhaus et al. 1956. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci* 1, 804 (1956), 801.

[Zhang et al.(2014)] R Zhang, Q Liu, and J Wei Chun-Gui. 2014. Huiyi-Ma,". In *Collaborative filtering for recommender systems," in 2014 Second International Conference on Advanced Cloud and Big Data*. 301–308.