RECOMMENDATION DIVERSITY WORTH CARING ABOUT

ABSTRACT

In recent years, human-human and human-computer interaction has been largely mediated by recommendation systems. The systems sometimes result in users forming echo chambers, where they only hear information they want to hear. They also result in homophily, where users of the same background are recommended similar things, resulting in greater political and social divides. We propose a set of two diversity metrics to specifically target echo chambers and homophily in recommendation systems. The first of these metrics uses a clustering approach on user embeddings. The second metric uses a set difference approach in order to determine how diverse the recommendations are, when taking into account a user's many different interests. We then attempt to improve upon previous diversity stimulating techniques by leaning on work in the evolutionary computation literature known as lexicase selection to pick the items that have a diverse yet high matching set of features to those that a user is interested in. We found that lexicase selection outperforms its ranked counterpart across modern recommendation systems, a variety of recommendation list sizes, and also four different metrics. This work highlights that lexicase selection or other techniques from the evolutionary computation literature are able to be applied to recommendation systems in order to help improve the diversity, move people away from echo chambers, and also help decrease the amount of homophily in the recommendations.

1 Introduction

Recommender systems are becoming a larger and larger part of out daily lives. They help us with our shopping, entertainment, research, and much more. These systems are at the heart of how we interact with technology, as well as how we interact with other people using technology. They allow us to find what we are looking for faster and predict what we want to see next. However, recommender systems are not perfect. In their mission for accuracy, they end up restricting our open mindedness. In their mission to be consistent, they end up being monotonous. These issues result in the disregarding of content that is exciting and interesting for users. Instead, they are shown content that is the same as what they're used to, or the same as what people similar to them like.

The extent to which we are shaped by the items and people we interact with has been significantly studied within the psychological and sociological literature [1, 2, 3]. Homophily, the idea that people that come from similar backgrounds tend to have similar interests and beliefs, is one such kind of shaping. Collaborative Filtering (CF) is built on principles very similar to homophily: it recommends items to you that similar users liked. These systems are responsible for shaping the content we view and chose to consume on a large scale. For this reason, it is important to determine whether these systems are contributing to negative cycles in social circles, and are not over-emphasizing echo chambers.

In this work, we present two different diversity benchmarks that are supported by the psychological and sociological literature. Different to the usual recommender system diversity measurements, our framework is directly set up to detect echo chambers and homophilous circles. We then present the application of a diversity preservation technique from evolutionary computation known as lexicase selection to improve the prevalence of items that have the form of instrumental diversity that we care about in this work.

2 Background and Related Work

There is much work in the recommendation system literature surrounding diversity of recommendations. Importantly, these systems most often use a form of dissimilarity to things that the user has seen before (novelty) or a dissimilarity

between things in the same recommendation list (diversity) to measure how much diversity a recommendation system is responsible for [4]. On the other hand, accuracy metrics in collaborative filtering and other recommendation approaches actually are based on similarity metrics: i.e. how much similar users liked this item, or how similar this item is to previously liked items [5]. [6] look at the list of recommended items as a single unit, and measure the intra-list similarity (ILS) to attempt to increase the diversity of the items shown to users.

[1] describes Homophily as the tendency of people that come from similar backgrounds to have similar likes and dislikes. The solution to this, called serendipity, is something that is hard to achieve. Many recommender systems attempt to reach this through a variety of diversity techniques, which are at a variety of levels of effectiveness.

There are many different ways to benchmark recommender systems. The most common methods are precision and recall. Precision measures the proportion of relevant items retrieved out of all the items that were retrieved. Recall, on the other hand, measures the proportion of the relevant items retrieved *out of all the items that are relevant to the user*. Both of these metrics measure a sense of "goodness" of recommendation [7]. The Hit rate of a recommendation system is the number of items that are predicted correctly to have been rated by the users in a held out dataset. There also exist some benchmarks that are beyond accuracy, or set to measure things that accuracy cannot. For example, coverage measures the proportion of the items that can have predictions generated for them. Personalisation measures if the system recommends different users similar items, which is simply the dissimilarity between different user's recommendation lists.

More relevant to this work are the benchmarks involving diversity of recommendation. The two main metrics in this area are those for "novelty", or how different an item is with respect to what a user has seen in the past, and "diversity", which is how different the items in the recommendation list are to each other [4]. The novelty metrics are often formulated as a surprisal model, where items that are less likely to be viewed have a higher surprise rating. These are usually counter-productive, as they show users items that are novel, but not at all interesting to them. The diversity metrics are often simply a inter-list similarity metric that measures how similar items are to the others in the recommendation lists. What these metrics leave out is any sense of user identity and embedding, which means that homophily cannot really be addressed by purely looking at inter-item similarity. In this work, we propose two new metrics that measure effective diversity, that will help detect echo chambers and serious cases of homophily.

We then propose a method inspired by the evolutionary computation literature to build a recommendation list that has this effective diversity. The use of features of evolutionary computation in the context of recommender systems is not a novel thing. [8] uses an evolutionary algorithm to effectively evolve recommendation lists that are both diverse and accurate by mixing an accuracy- and diversity-focused recommendation algorithm. [9] uses Multi-Objective Evolutionary Programming to optimize for the conflicting objectives of accuracy and diversity at the same time. For a more general review of evolutionary computation in recommender systems, please see [10]. The main difference between our approach in this paper and those in previous work is that we do not perform an entire evolutionary loop. Instead of selection followed by reproduction for a number of generations, we only perform the selection step in order to filter down a short list of candidate items to a set of recommendations.

We begin by a discussion of detecting echo chambers and homophily, and then move on to discussing a proposed method to help mediate these issues. Finally, we end by benchmarking this system on one of our diversity metrics, and discuss the results.

3 Detecting Echo Chambers and Homophily

Often, the RS literature uses the inverse of preferences to correlate with diversity metrics. We define effective diversity to be the kind of diversity that is needed in order to promote users leaving certain echo chambers, and becoming more serendipidous. In order to determine whether recommender systems are indeed promoting effective diversity in their recommendations, new benchmarks need to be developed. We discuss two different methods to measure the extent to which recommender systems amplify the existence of echo chambers and homophily.

The first method, a graph based approach, determines clusters of users and finds users that are significantly similar to each other. The second method, working purely on the user and item embeddings, finds extent to which the recomender system shows similar users similar items.

3.1 Embedding Clustering Approach

In order to measure echo chambers we need to take into account two different aspects of every recommender system: the content being recommended (henceforth referred to as *items*) and the *users* being recommended to. Echo Chambers are a phenomenon that involves both of these components in different ways. The items that a recommender system

suggests are pieces of content that the recommender system has identified the relevance of using various techniques such as CF, this means that it has an internal representation of each piece of content in what can be considered *item space* (all possible item locations in the dimension of item encodings). As a result, we are able to extract at any given state of the recommender system, all of the information that it encodes about a given item. We can then - after the cold start has been given time to "warm up" - take a look at all n items that were recommended to a specific user. This can be done by considering the current recommendation as $t = 0$ and taking a look at how each item in the past differs from the currently recommended item:

$$\texttt{SelfSim}_{I_k} = \sum_{j=-n}^{1} d(I_k^j, I_k^0)\lambda^{-j} = \sum_{j=1}^{n} d(I_k^{-j}, I_k^0)\lambda^j$$

Where $\texttt{SelfSim}_{I_k}$ is the self similarity score of user k's recommended items where I_k^j is the per-user best item by acknowledgement (ie; by positive interaction, click, or other metric) at time $t = j; j \in [-n, 0)$. d is a distance function and $\lambda \in (0, 1)$ is a decay factor over time. The `SelfSim` score for each user is then used as a metric for how much the particular user has changed their content preferences over time; we posit that a user that has been in an echo chamber - by definition - does not change the content they view and rate highly as much as other users over time [11].

Now this by itself is not enough to detect an echo chamber, because a user's echo chamber has a another component: the tight community structure.

> "The higher the controversy between members of different groups and the homogeneity between members of the same group, the higher the probability to be in the presence of echo chambers." [2]

Therefore this community structure has two main characteristics:

A. The communities are 'tightly knit'
B. The communities 'repel' each other

We address these characteristics using a two step process. In the first step we address **A: The communities are tightly knit**. We do this using the recommender system's user space (all possible user locations in the dimension of user encodings) and the function below:

$$\texttt{PScore}_k = \sum_{i \in N; i \neq k} \frac{1}{\frac{1}{D_{\max}} + d(U_k, U_i)^{1+\gamma}}$$

Where $\texttt{PScore}_k$ is the proximity score of user k in user space. This is a sum over the reciprocal distance of all users except for the one being scored which means that two users that are closer together are given an asymptotically better proximity score than ones further apart - whose `PScores` decay to zero. N is the number of users in the user space. d is a distance function being raised to $1 + \gamma$ which is one more than a proximity discount factor which is a hyperparameter (positive γ). $D_{\max}$ is the max density (as $d \to 0$ then this becomes the inter-user summed term) - this is also a hyperparameter. Do this for every term and keep track of values for all $k \in [0, N]$.

Now to address the second characteristic **B: The communities repel each other** we use another process. For a community to repel all other communities, we abstract all communities as clusterings of users in user-space, and we abstract their repellent nature as inter-cluster centroid distance. In order to create clusterings - for which we use the KMeans algorithm - we need to define the number of clusters to create. In order to do that we propose this algorithm:

A. Create a linearly spaced array of values $h \in [h_i, h_e)$ (where h_i can be around 20 and the granularity can be around $\frac{1}{50}$).
B. Use the cluster decay function: $f(x) = (1 - x)e^{-hx}$.
C. With this function we create circular cluster boundaries with radii (R) equal to

$$R_{U_k} = \frac{1}{N} \sum_{m=0; m \neq k}^{N} f(d(U_k, U_m))$$

   in user order from highest $\texttt{PScore}_k$ to lowest.
D. At each boundary creation we exclude any users that are within a boundary radius.

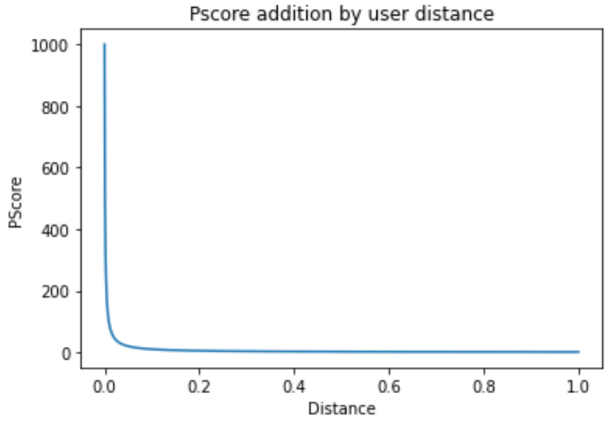

Figure 1: Each user's PScore is updated with respect to every other user's distance. Here $D_{\max} = 1000; \gamma = 0$.

E. Repeat steps **B** through **D** until there are no more points left outside of a boundary. The number of circular boundaries is the candidate for the number of possible clusters at h.

F. Repeat the entire algorithm for each value of h and save all of the possible cluster numbers.

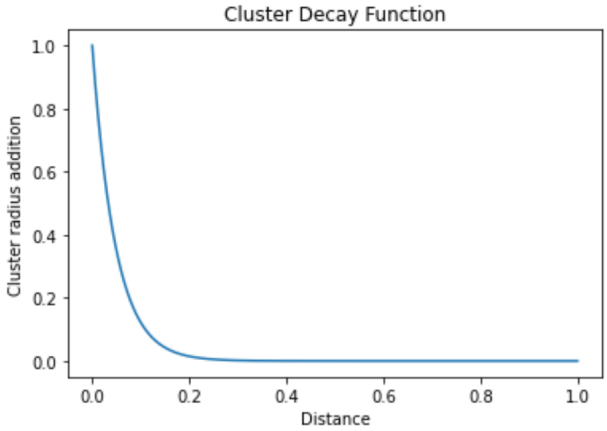

Figure 2: This is the cluster decay function with $h = 20$

Once that algorithm finishes running, use the most common cluster numbering as the parameter for KMeans. Now that we have the KMeans clustering, what's left is to propagate a "repellent score" within clusters that have repelled others and so we assume to be far away. We so this using a simple algorithm defined below:

A. Compute the average distance between every centroid and every other centroid: $TAD = \frac{1}{|C|^2} \sum_{v \neq b} d(C_v, C_b)$.

B. Compute the individual average centroid distances $CAD = \frac{1}{|C|} \sum_b d(C_i, C_b)$ and σ the standard deviation of the CADs

C. If the individual average centroid distance is less than the total average centroid distance then give it an $\texttt{RScore}_k$ of 0.

D. Otherwise for every point in that cluster that is within the total average centroid distance relative to the centroid, give it a score $\texttt{RScore}_k$ defined as:

$$\texttt{RScore}_k = N(1 + NCAD)^2(1 - CD)^2$$

where CD is the user's **c**urrent **d**istance away from the centroid and $NCAD$ is the **n**ormalized **c**urrent **a**verage **d**istance (the normalized individual average distance) defined as $\frac{CAD - TAD}{\sigma}$.

Now to put all of these together we compose the three parts together. First we compute $\texttt{SelfSim}_k$ for all users and take the top N using that metric. Now we restrict user-space to only those users, scale each axis to within $[0, 1]$, choose

appropriate hyperparameters, and apply the community detection characteristics onto them. Once this is done we finally compute $\texttt{UScore}_k = \texttt{PScore}_k + \texttt{RScore}_k$ which gives us a ranking of *echo chamber susceptibility* for each user in the restricted user-space.

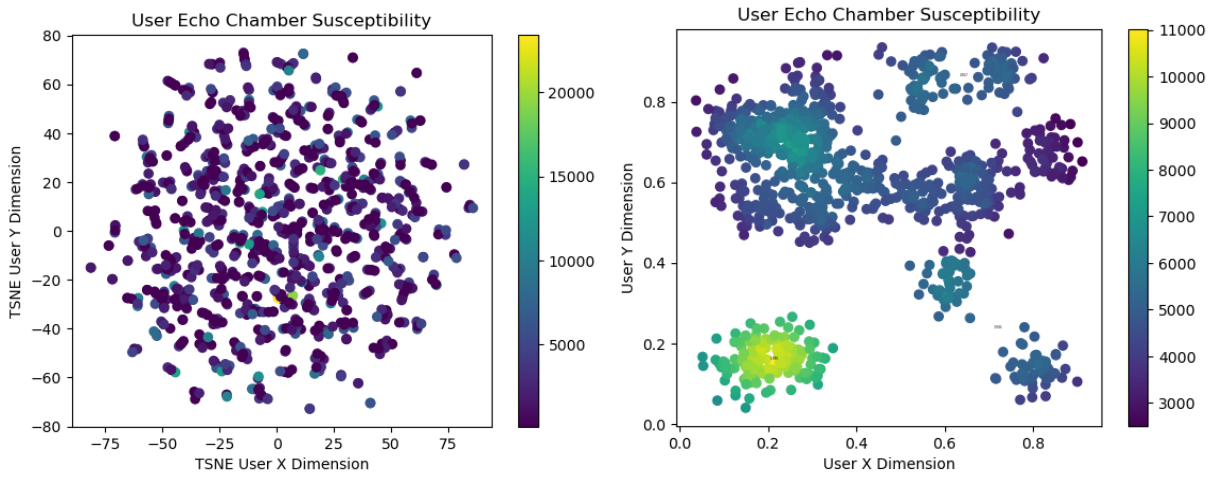

Figure 3: MovieLens User Embeddings Echo Chamber Susceptibility (embeddings generated using neural matrix factorization) (left). Idealized User Embeddings Echo Chamber Susceptibility (embeddings generated using SciKit Learn's *makeblobs*) (right). Brighter is more susceptible.

3.1.1 Explanation of Results

In the left graphic we have mostly dark, sparse clusters with small portions of susceptible users. This is indeed what we are looking for because we need a balance of density combined with how far away these tiny clusters push others away. It is easy to see that clusters with highlighted points are the densest and have the largest blank rings around them, or are less dense but also in a remote corner of the space.

In the idealized graphic we have a clearer image of what the algorithm does, we know that the lower left-hand cluster is both very dense and very isolated from the rest of the users and so it is most highly rated for susceptibility. The lower right-hand-side cluster is sparser and closer and so is ranked lower even though it is further out. The code used to generate these figures has been hosted online [1].

The time based nature of this metric requires that we have future data to evaluate the recommendation systems. As the data that we have was all created using the same recommendation system, it is not fair to *look backwards* and use these values to measure the presence of homophily or echo chambers. Instead, to use this metric in practice would require a real world deployment, or a simluation. Both of these approaches are out of the scope of this investigation. In order to address this issue, we present a second metric that does not require a time dimension.

3.2 Set Difference Approach

In order to devise a metric that can be tested on recommendation systems with a static dataset, we propose a method that simply tracks the differences in the sets of different recommended items. The above benchmark analyses the extent to which a recommender system is pushing people into echo chambers, or promoting homophily by encouraging people that are similar to consume similar content. It did this by tracking the values of the user and item embeddings over time. We now present a less graphical, and more mathematical, approach to achieve the same thing. This method does not require a time variable, which makes it applicable for recommender systems that do not have this information.

3.2.1 Diversity Metric

The general idea powering this metric is the fact that a diverse list of recommendations isn't particularly useful if the individual for whom they are recommended is completely disinterested in everything on the list. Therefore, to start, we make clusters out of movie embeddings and only consider the clusters deemed relevant to the user. Relevance is

[1] Link to Google Collab Notebook

determined using user history as follows. If a user has rated an item high (above a certain threshold) in some cluster then that cluster is relevant. Furthermore, any cluster that is nearby (determined by clustering on top of clustering) said cluster is also deemed relevant. We will call this collection of relevant clusters C. We will call U the number of relevant **unique** clusters represented by the items in R. Where R is the list of recommendations generated by some recommender system for a particular user. We will calculate our metric as follows:

$$D = 1 - \frac{\min(|C|, |R|) - U}{\min(|C|, |R|)}$$

3.2.2 Intuitive Understanding

The metric will be high when the total number of relevant clusters and the number of relevant clusters represented in the recommendations are close. That is, for diversity we want to ensure as many relevant clusters of items are represented as possible. It's also worth noting that we use $\min(|C|, |R|)$ in place of $|C|$ as there could be more relevant clusters than the number of recommendations we are testing the metric on. So to make the scores meaningful by taking a minimum of both of these numbers, as the maximum number of relevant clusters you can recommend from is bounded by number of recommendations, if $|R| < |C|$, and in other cases we are bounded by number of relevant clusters. We can also observe that the metric reduces to

$$1 - \frac{\min(|C|, |R|) - U}{\min(|C|, |R|)} = 1 - (1 - \frac{U}{\min(|C|, |R|)}) = \frac{U}{\min(|C|, |R|)}$$

This makes sense because the higher the number of unique and relevant clusters represented the better (more diverse) and closer to 1 we will be. While the fewer unique and relevant clusters represented the closer to 0 (and less diverse) we will be.

EXAMPLES:

Say,

$R = \{I_1, I_3, I_5, I_4, I_7, I_{11}\}$, $|R| = 6$, where I_i is item i, R is our list of recommended items

$C = \{C_1, C_2, C_4\}$, $|C| = 3$, where C is our relevant clusters

Highest Diversity Example:

Say clusters were found to be,

Cluster 1: $\{I_1, I_{11}\}$, Cluster 2: $\{I_5, I_7\}$, Cluster 3: $\{\}$, Cluster 4: $\{I_3, I_4\}$

$D = \frac{3}{\min(3,6)} = 1 = 1$

Mid-Tier Diversity example:

Say clusters were found to be,

Cluster 1: $\{I_1, I_4, I_{11}\}$, Cluster 2: $\{I_5, I_7\}$, Cluster 3: $\{I_3\}$, Cluster 4: $\{\}$

$D = \frac{2}{\min(3,6)} = \frac{2}{3} = \frac{2}{3}$

Poor Diversity example:

Say clusters were found to be,

Cluster 1: $\{I_1, I_3, I_{11}, I_5, I_4\}$, Cluster 2: $\{\}$, Cluster 3: $\{I_7\}$, Cluster 4: $\{\}$

$D = \frac{1}{\min(3,6)} = \frac{1}{3} = \frac{1}{3}$

Lowest Diversity example:

Say clusters were found to be,

Cluster 1: $\{\}$, Cluster 2: $\{\}$, Cluster 3: $\{I_1, I_3, I_{11}, I_5, I_4, I_7\}$, Cluster 4: $\{\}$

$D = \frac{0}{\min(3,6)} = \frac{0}{3} = 0$

Note: The final case is when, since relevant clusters are found by looking at user history and not recommendations, all recommended items are from “irrelevant” clusters.

3.3 Explanation of Results

This approach is a simple way to track the extent to which items being recommended to a user are “spread out" across their interests. Instead of having the entire recommendation list taken up by a single cluster of items (which could be a sign of echo chambers), recommendation lists with effective diversity are those that have a wide range of relevant items to the user.

4 Lexicase Selection for Recommendation Systems

Now that we have determined what homophily or an echo chamber might look like, we will discuss methods to tackle such things in the context of making better recommendations.

The preservation of effective diversity requires more than simply “novel" items. Truly diverse items do not fit the pattern of what the user has seen in the past. This does not necessarily imply that the best way to select items to tackle echo chambers is to show users items that they are predicted to hate. Indeed, these recommendations do not fit the pattern of what the user has seen in the past, yet they seem to be missing something crucial: a reason to care. To provide this reason, we must find items that the user has a partial interest in, yet is different to what they normally watch. Oftentimes, these items are lost by the recommender system’s preliminary filtering due to their dissimilarity to most items that the user has interacted with. To ensure that these items are not lost from the list, a mechanism to preserve the items that have effective diversity needs to be employed.

In order to preserve this kind of diversity, we can take a page out of the evolutionary computation diversity preservation literature. In evolutionary computation, an individual is a candidate solution for the problem being solved. Many individuals are generated randomly, evaluated, and then selected from to find the top individuals, who will become the parents of the next generation [12]. The parents are then mutated and crossed over to create children, who are then evaluated and selected from sequentially. This loop continues until an individual is found that exhibits the expected behavior. Diversity in these systems is largely important, as each solution has the chance of leading to a better solution in the future, so more diversity means more potential pathways to success.

Diversity is promoted in many different ways in evolutionary computation. Some preliminary methods, like novelty search [13], purely focused on the novelty of an individual, and disregards completely any metric of performance. Later work combined this with quality metrics to form the field of Quality Diversity (QD) [14, 15, 16]. The QD approaches define “behavioral descriptors" (BDs) that act as defining metrics for the behavior of a certain candidate solution. Note here that “behavior" is hard to define for a static recommendation, and so applying QD to recommendation systems would require significant human insight.

In order to avoid having to define behavioral (or diversity-specific) metrics, we must find a diversity preservation technique that does not rely on these explicit diversity clues. Lexicase selection [17, 18] is a parent selection method that has been shown to be effective at increasing diversity in the evolutionary computation literature [19, 20]. Importantly, it promotes this diversity without explicitly being tasked to do so, as it only acts on de-aggregated scores on different performance metrics. These performance metrics are much easier to procure in the recommendation system environment as these are usually already defined for us.

In practice, using Lexicase selection means we do not require a significant amount of human insight into defining diversity metrics that can be used to explicitely select items to include in the recommendation list. Instead, we use the matching of a user embedding to a specific dimension of item space. As opposed to taking the dot product between a user and item embedding, we instead find the hamming distance between these as a metric of how much this item matches a user’s interests on a specific dimension.

A concrete example of this is the following:

Say we have an item and user embedding that is generated by some collaborative filtering technique such as matrix factorization (MF). The item and user embeddings could be set up as vectors of dimension 3 (for the example), where the dimensions represent abstractly “comedy", “action" and “romance" respectively.

$$u_1 = \begin{bmatrix} 1 \\ 0 \\ 0.8 \end{bmatrix}, I_1 = \begin{bmatrix} 0.9 \\ 0.8 \\ 0.1 \end{bmatrix}, I_2 = \begin{bmatrix} 0 \\ 0 \\ 0.8 \end{bmatrix}, I_3 = \begin{bmatrix} 2 \\ 0.2 \\ 0.7 \end{bmatrix}$$

Data:

- `features` - a sequence of all features (dimensions) of each item to be used for selection (higher is better for a user)
- `items` - the (possibly shortlisted) list of items to be selected from

Result:

- an item to add to the recommendation list

for *feature in shuffled_features* **do**
  $i \leftarrow$ the index of `feature` in `features`
  `items` $\leftarrow$ the subset of the current `items` that has within ϵ of the best match for this feature.
  if *items contains only one single item* **then**
    return *item*
  end
end
`item` $\leftarrow$ a randomly selected item in `items`
return *item*

Algorithm 1: Lexicase Selection for Item Selection

The prediction of how much user 1 will like item j is simply computed as $u_1 \cdot I_j$. For example, user 1's predicted rating for item 1 is 0.98. Instead of aggregating to a single number, lexicase selection requires a vector of perceived scores. So, we instead use an elementwise multiplication of the user embedding vector with the item embedding vector as follows: $p_{i,j} \in \mathbb{R}^3 = u_i \otimes I_j$. Notice that the previous score is simply the sum of the values in this vector.

Explained above is one such way to *de-aggregate* the matching of a single user to a single item. There are many more. The key thing here is that by deaggregating the preference of the user, we can determine items that match the user's preferences *in unique ways*. With lexicase selection, items that, using a regular recommender system, would have a low predicted score, could still be selected if there is a particular feature that the user is extremely interested in.

What is important to note is that we can use *any* de-aggregation that we would like to present the multiple dimensions of "liking" that we expect users to have. In this work, we use the second-to-last layer values of a neural network used for collaborative filtering and NeuMF to generate the recommendations that are diverse, and importantly so.

In most applications of lexicase selection, the set of items is filtered down by only keeping the items that have *exactly* the best score on each dimension (match on a feature). Due to the fact that our features are often real-valued, it is not appropriate to take the items at each step that have the best possible matching on that feature, as there will likely never be any ties (leading to only one feature ever being used for selection). As we often want interestingly unique combinations of features, we must relax the elitness condition that is used in each filtering step. To solve this issue, we adopt a version of lexicase selection that works for real-valued fitness functions known as epsilon lexicase selection [21].

The algorithm for lexicase selection for recommendation systems can be found in Algorithm 1[2] (adapted from [22]). First, we begin by determining the so-called preference features (the deaggregation of the single preference value that a user has for an item) for every item for a single user. To build a recommendation list for this user, one begins by shuffling the indices of the features into a random order. This ordering will be user to select a single item to be added to a recommendation list. For every feature index in the list, in the order that they were shuffled into, we begin to filter down the set of candidate items by only keeping the items that have within ϵ of the best match for the user *for that feature*. Note that the item's match on all other features is disregarded at this step. This algorithm has the run time of $\mathcal{O}(N \cdot C)$ where N is the number of features, and C is the number of items to be selected from. This runtime is simply to select a single item. Whilst this seems to be a large number, [23] shows that in practice, the number is much lower. Despite this, the runtime of lexicase selection still seems to be one of its major disadvantages.

What this means, given our simple example above, is the following. Say that the first feature in an arbitrary shuffle is feature 3, or romance. Item 2 has the highest score at that dimension (as $0.8 * 0.8 > 0.8 * 0.1$ and $0.8 * 0.8 > 0.8 * 0.7$), so it will be selected to be shown to the user. Notice that item 2 has a very low match on feature 1, or comedy. are that

[2]We implemented a variant of this algorithm for computational efficiency, where we only run the outer loop a set number of times rather than running it for all features in the shuffled list. If you chose the set number high enough you enjoy a gain in computation with a loss of the benefits that come with full lexicase. Through a set of preliminary experiments, we found 10 to be a good bound when the number of features was within 100.

if a user likes romance, and this given movie has a higher score than all others for romance, regardless of how much comedy it has or action, it could be recommended to the user.

5 Benchmarking Recommender Systems

Our experiments were performed using two different recommendation system architectures, Neural Collaborative filtering (NCF) and Neural Matrix factorization (NeuMF) [24]. For both systems, we compared performing lexicase selection on a deaggregated preference score for a user-item pair, to simply using the aggregated preference score to select these items. We also compare a mixing method, that mixes both lexicase selection and the baseline together.

Hyperparameter tuning was done using a random search on all hyperparameters, and training model on 70% of the data and testing it on rest using Mean Absolute Error as the performance evaluation of the model.

We compare the performance of different final list creation methods across both NCF and NeuMF for making top-5, 10, 20, 50, and 100 recommendations. Here top-k recommendation is just selecting k best items for the user using the appropriate method as described earlier. We use four different metrics to evaluate the performance of each of our recommendation systems: Coverage, Personalization, Hit Rate, and our above proposed set difference metric[3]. We compare each of NCF and NeuMF with rankings (r), lexicase (l), or a mix of both (m). The mix was conducted by a simple interleaving of a list generated with rankings, and a list generated with lexicase. With rankings, items are added to the top-k recommendation list based simply on the aggregated predicted score. With lexicase, items are added based on the lexicase selection on a deaggregated list of predicted score features. This list was simply generated by taking the second to last (where the output is the last) layer values, multiplied by the sign of the weights leading to the output.

The code used for these experiments is hosted online [4].

6 Results and Discussion

The results for Coverage, Personalization, Hit Rate and our proposed Diversity Metric can be found in Figures 4, 5, 6 and 7, respectively. We can see that lexicase selection or a mixed method outperforms the ranking based-approach using all four of the metrics. We will now discuss each metric in turn.

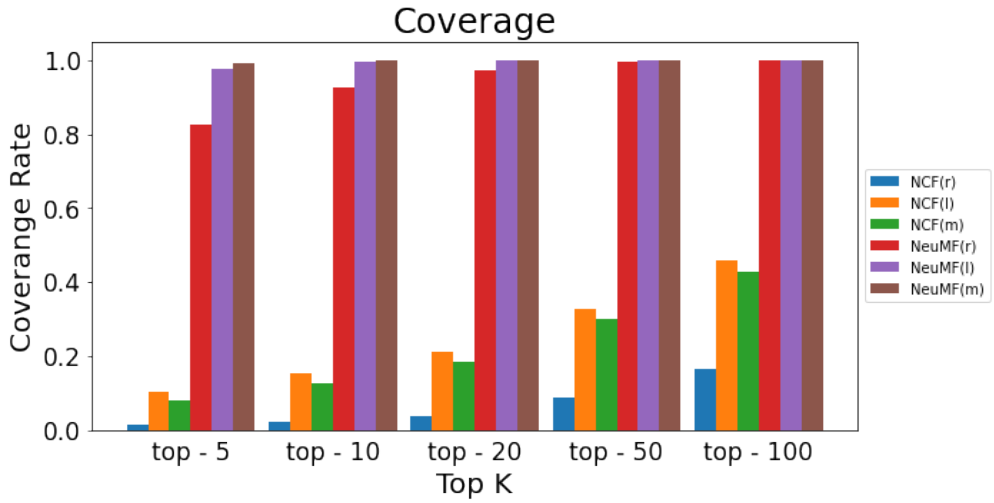

Figure 4: Coverage for Top-K recommendations for a variety of values of k, for a set of recommendation systems.

The coverage metric (Figure 4) measures the proportion of the items that are recommended for at least a single user, the higher the better. We see two main conclusions from this figure. Firstly, that NeuMF vastly outperforms NCF across all configurations. Secondly, we see that the variants using lexicase selection perform better than the respective

[3]We did not include a benchmarking of recommendation systems using our embedding clustering approach due to the fact that it requires a time dimension. In order to generate this data, we would either need to test a recommendation system in real time, or synthesize data, which are both out of the scope of this investigation.

[4]Google Colab Link for Experiment code

versions using a ranking-based approach. This could be likely due to lexicase selection helping select items that have high matching in some dimensions, but are low matching in others, and such would not be selected if you considered an aggregate measure. When using a de-aggregated measure, they are selected more often, which could explain the increase in coverage.

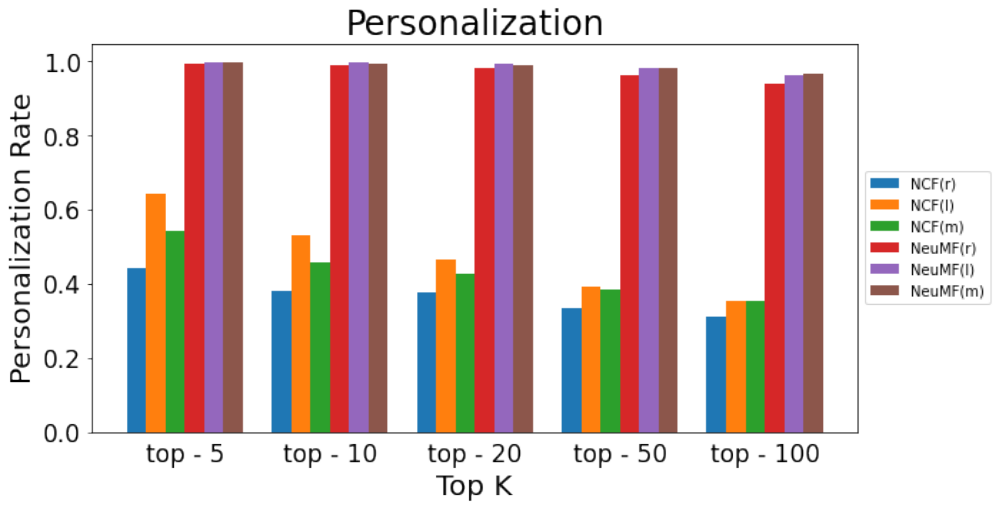

Figure 5: Personalisation for Top-K recommendations for a variety of values of k, for a set of recommendation systems.

The second metric analyzed was personalisation (Figure 5). This is the metric that measures how different the recommendations are for each user, the higher the better as we want each user's recommendations to be not related to one another, hence avoiding echo chambers. We can see here NeuMF once again outperforms NCF, and that lexicase selection also outperforms the ranking based approach. This could be due to lexicase selection's ability to specifically target features that have a high degree of matching for a specific user, without regarding the features that the user might not like. This could lead to more personalized recommendations as users are shown things that are more specifically tailored to their interests in interesting and unique ways.

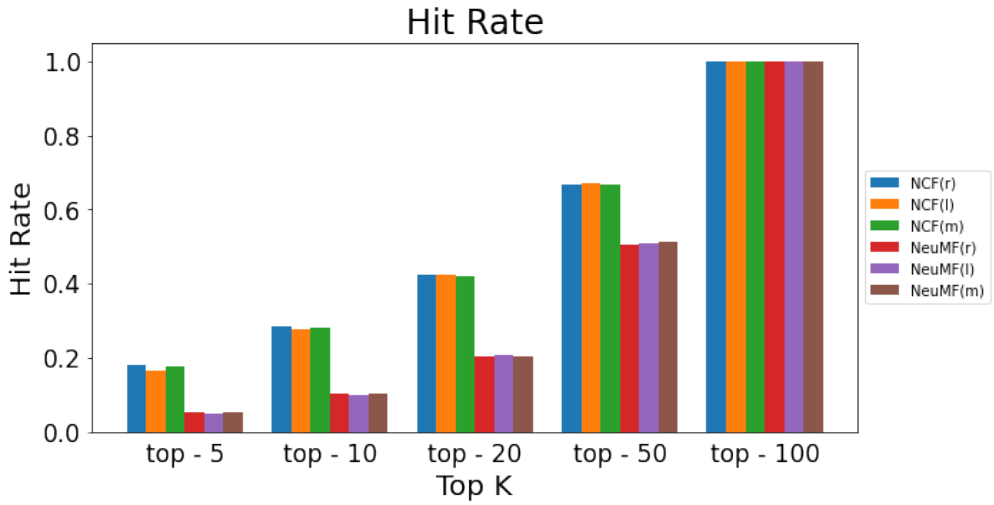

Figure 6: Hit Rate for Top-K recommendations for a variety of values of k, for a set of recommendation systems.

The third metric analyzed was Hit Rate (Figure 6). This metric measures the ability of a recommender system to accurately predict which items a user also has rated in the dataset. For this metric, as we would expect, we see that all of the three different methods (ranking, lexicase or mix) perform relatively equivalently. This is the expected result as we are utilizing lexicase selection to help preserve the diversity of the recommendations whilst not really attempting to

improve the hit rate. Interestingly, we also found that the NCF methods often had a higher Hit Rate than those using NeuMF.

It is also important to note that metrics like hit rate are used to measure how well the recommendation system does at predicting items that users have interacted with *in the data set*. This data set might have been generated while using a separate recommendation system, and so the results using hit rate and similar metrics are biased towards recommendation systems that are similar to those used to generate the original data set. With our lexicase selection based systems, the recommendations perhaps do not align with the data set, but could indeed still have a higher hit rate.

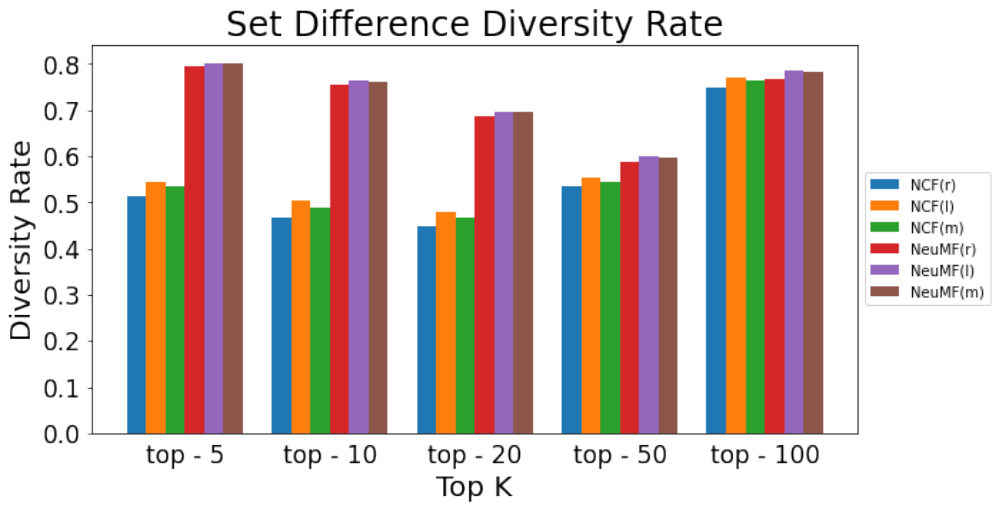

Figure 7: Set difference diversity (our proposed metric) for Top-K recommendations for a variety of values of k, for a set of recommendation systems.

The final metric used is our own diversity metric using set differences (Figure 7). We can see here that the approach using lexicase selection does indeed outperform those using ranking, however the magnitude of this improvement was lower than we expected. This was a surprising result as lexicase selection was specifically used in order to generate larger amounts of diversity as measured by the metric we proposed. Despite this, there is a consistent improvement over NCF with rankings and NeuMF with rankings.

A reason for the result metrics that involve measuring diversity, such as our proposed metric, lose information when averaging across all users in the dataset. It is often the case that most users on a recommendation platform are indeed not in an echo chamber, and so have relatively high diversity. This would mean that any method of generating a recommendation list could have a relatively high diversity when using our metric. This hypothesis could also explain the result that the difference in performance between NCF and NeuMF on these metrics are not as large as for other metrics. It could also help explain why the improvement with using lexicase selection is not very large for our proposed diversity metric.

7 Conclusion

We proposed a set of two diversity metrics to specifically target echo chambers and homophily in recommendation systems. The first of these metrics uses a clustering approach on user embeddings. These embeddings are tracked over time in order to determine whether or not they are moving towards or away from an echo chamber. Due to the fact that this diversity metric is difficult to use in practice (as it requires future forcasting, or real-world study), we propose a second diversity metric. This metric uses a set difference approach in order to determine how diverse the recommendations are, when taking into account a user's many different interests.

We then proposed an augmentation to a recommendation system based on diversity preservation work in evolutionary computation. We propose to use a de-aggregated form of preference prediction, which is predicting preference for certain item *features* instead of items themselves. Then, we use a form of selection known as Lexicase selection to pick the items that have a diverse yet high matching set of features to those that a user is interested in.

We compared two different recommendation systems, Neural Collaborative Filtering and Neural Matrix Factorization with regular aggregated ranking, de-aggregated lexicase selection, and also a mix of the two. We found that lexicase

selection outperforms the ranked counterpart across both systems, a variety of recommendation list sizes, and also four different metrics. The mixed method occasionally also outperforms lexicase selection.

This work highlights that lexicase selection or other diversity preservation techniques from the evolutionary computation literature are able to be applied to recommendation systems in order to help improve the diversity, move people away from echo chambers, and also help decrease the amount of homophily in the recommendations.

This work opens many different pathways for future research. The first is that we would like to see a benchmarking of systems using lexicase selection using our proposed embedding clustering approach. Although it might be more difficult to use, we predict that it will be a more faithful detector of echo chambers being formed in real time. Another possible area of study is to directly address homophily by using lexicase selection on the user demographic context, as opposed to just their embeddings. This will help lexicase selection select items that are diverse for a person *given* who they are and their background. Future investigation would also benefit from using lexicase selection to its full capacity, as opposed to limiting to just 10 features at most being used for selection. Finally, the application of other diversity preservation techniques from the evolutionary computation literature, such as Quality Diversity, would be a very interesting area to move into.

References

[1] Ethan Zuckerman. Homophily, serendipity, xenophilia, Apr 2008.

[2] Marco Viviani Giacommo Villa, Gabriella Passi. Echo chamber detection and analysis. *Social Network Analysis and Mining*, 11(78), 2021.

[3] Bart P. Knijnenburg, Saadhika Sivakumar, and Daricia Wilkinson. Recommender Systems for Self-Actualization. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 11–14, Boston Massachusetts USA, September 2016. ACM.

[4] Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and Diversity Metrics for Recommender Systems: Choice, Discovery and Relevance. In *In Proceedings of International Workshop on Diversity in Document Retrieval (DDR*, pages 29–37, 2011.

[5] Mi Zhang and Neil Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 123–130, New York, NY, USA, October 2008. Association for Computing Machinery.

[6] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web - WWW '05*, page 22, Chiba, Japan, 2005. ACM Press.

[7] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, January 2004.

[8] Zhipeng Hou and Jing Liu. A Two-phase Evolutionary Algorithm for Solving the Accuracy-diversity Dilemma in Recommendation. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE.

[9] Edward Hinojosa-Cardenas, Edgar Sarmiento-Calisaya, Cesar A., Lehi Quincho-Mamani, and Jair F. Multi-Objective Evolutionary Programming for Developing Recommender Systems based on Collaborative Filtering. *International Journal of Advanced Computer Science and Applications*, 11(10), 2020.

[10] Tomas Horvath and Andre de Carvalho. Evolutionary computing in recommender systems: a review of recent research. *Natural Computing*, 16, 09 2017.

[11] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. Temporal diversity in recommender systems. Geneva Switzerland, July 2010. ACM.

[12] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Complex adaptive systems. MIT Press, Cambridge, Mass, 1st mit press ed edition, 1992.

[13] Joel Lehman and Kenneth O. Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19(2):189–223, June 2011.

[14] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[15] Justin K Pugh, LB Soros, Paul A Szerlip, and Kenneth O Stanley. Confronting the challenge of quality diversity. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 967–974. ACM, 2015.

[16] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[17] Lee Spector. Assessment of Problem Modality by Differential Performance of Lexicase Selection in Genetic Programming: A Preliminary Report. 2012.

[18] Thomas Helmuth, Edward Pantridge, and Lee Spector. Lexicase selection of specialists. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1030–1038, Prague Czech Republic, July 2019. ACM.

[19] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. Effects of Lexicase and Tournament Selection on Diversity Recovery and Maintenance. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 983–990, New York, NY, USA, July 2016. Association for Computing Machinery.

[20] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. Lexicase Selection for Program Synthesis: A Diversity Analysis. In Rick Riolo, W.P. Worzel, Mark Kotanchek, and Arthur Kordon, editors, *Genetic Programming Theory and Practice XIII*, pages 151–167. Springer International Publishing, Cham, 2016. Series Title: Genetic and Evolutionary Computation.

[21] William G. La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase selection for regression. *CoRR*, abs/1905.13266, 2019.

[22] Li Ding, Ryan Boldi, Thomas Helmuth, and Lee Spector. Lexicase selection at scale. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2054–2062, 2022.

[23] Thomas Helmuth, Johannes Lengler, and William La Cava. Population Diversity Leads to Short Running Times of Lexicase Selection, April 2022. arXiv:2204.06461 [cs].

[24] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural Collaborative Filtering, August 2017. arXiv:1708.05031 [cs].