Neighborhood Construction through Item Popularity in Collaborative Methods

A study on the impact of items popularity on recommender systems

Diogo Lima Nicolau
Instituto Superior Tecnico
Lisbon, Portugal
diogo.lima.nicolau@gmail.com

ABSTRACT
A Recommender System seeks to select and recommend custom items based on user interests and context. Collaborative algorithms used in recommendation systems have acquired great importance, mainly due to their use in electronic commerce, as is the case of Amazon. This work pursues to study the impact of different neighborhood strategies in the formulation of recommendations through collaborative methods. As a basis for this study a state of the art analysis in recommender systems is carried out as the analysis of the most appropriate evaluation metrics. Several popularity strategies are studied and implemented using the framework Lenskit. The aim is to show the usefulness of popularity as a significant indication in the creation of recommendations. However to achieve this propose we seek to interpret the results through an additional set of techniques and assess the impact of the implemented strategies over the long tail.

CCS CONCEPTS
• Human-centered computing → Collaborative filtering; • Information systems → Recommender systems; Information retrieval diversity; Novelty in information retrieval; Relevance assessment;

KEYWORDS
similarities; neighborhoods; predictions; preferences; popularity; collaborative methods

1 INTRODUCTION
Good recommender systems can increase sales by several percentage points. The success of large online platforms, such as Amazon [1] and Ebay [12], are achieved through the quality of their recommendation algorithms [9]. These platforms have a large number of users and products sold. Gathering information, such as user feedback, for all products is an impossible task. The percentage of products duly classified by users is in fact very short. This sparsity problem makes it difficult to create useful and personalized recommendations. It is necessary not only to create solutions capable of scaling with the increase of the information generated by the users, but also, to avoid the resulting noise from so much information.

The focus of this project becomes the study and implementation of new building techniques for item neighborhoods. The neighborhood is a set of relevant elements that allows for a given element the prediction of its classification. Nevertheless it is not immediate what elements to choose to constitute a neighborhood. In certain cases there is detailed information about the elements’ features, allowing the neighborhoods construction through these features. But in cases of collaborative content there is only the transactional information between two parties. The choice of which transactions are used is governed by a heuristic. This heuristic must determine how many and what elements will constitute the neighborhood. The inherent problem of this project is the need to establish heuristics that minimize the amount of information used in collaborative environments. That is, for each element we want to find the minimum set of elements that establish a neighborhood that in turn results in legitimate and coherent recommendations.

The famous saying - “Tell me who you walk with and I’ll tell you who you are.” summarizes the purpose of the neighborhood. In particular, we try to understand what happens when neighborhoods are created through heuristics based on popularity, that is, what behavior does the element presents us with if it “hangs around” with the “popular” ones for instance. By working directly with the concept of popularity we expect to deal with the long tail. The long tail term interprets a new and revolutionary trend of the future in the business world where “the future of business is to sell less than more products”[6]. By understanding the “less known” items we can work with them and provide recommendations that are not only less biased towards the “most popular” items but seize every opportunity to present this set of items.

In this project we make an important assumption. Usually projects related to neighborhood selection focus on the prediction moment. However this time we focus on the similarities computation moment. Following a similarity function the system must compute the similarity between items before guessing the classification an user $p$ gives to an item $i$. Commonly the predictor makes use of all the similarities and selects a set of items as neighbors for $i$. The most $N$ similar items to $i$ are chosen and the weighted average of the classifications given by $p$ to the selected set of items is calculated. To avoid the computation of meaningless similarities we focus our neighborhood study a step back. If it becomes possible to know $a priori$ which set of items provides good enough similarities to the predictor we bypass a great amount of work. Therefore we seek to create neighborhood strategies for the similarity computation moment and ascertain which items similarities should be calculated. If it’s the case that the elected strategies actually capture an user behavior pattern we can not only reduce the amount of work but also the amount of noise digested during all the process.
2 RELATED WORK
In the article [2], Jon Herlocker, Joseph A. Konstan and J. Riedl describe the lack of consensus on which techniques are most appropriate for each situation in collaborative filtering systems. As well as the lack of studies on how significant the impact of each parameter on system accuracy is. Consequently, when implementing a collaborative filtering system there is a set of decisions difficult to take without any guidance. The authors propose a guide to facilitate these decisions, dividing the system into three parts: computation of similarities, selection of neighborhood and combination of classifications. The second component - neighborhood selection - is the one with the greatest interest in the universe of this study. In the article the authors present several ways of weighing the contribution of each user for the prediction calculation, making use of the distance between users where distant users contribute less than those whom are closer. Although the user-based context is not the one followed in this project, the conclusions are very useful: commercial applications rely on millions of users so doing this kind of calculations becomes impractical in real time. The system then has to select a community. The article suggests two distinct ways of calculating neighborhoods in prediction calculations: Correlation weight threshold and Maximum number of neighbors used.

In the article [3] the authors present a set of results about the use of a random strategy for neighborhood choice. When compared to other approaches such as clustering based collaborative filtering, the authors show how the random choice is surprisingly better at predicting results. This fact weighs even more since the random choice results in a neighborhood in the order of tens, against the use of all items as neighborhood. The study of this article culminated in the implementation of this neighborhood strategy as part of the solution, described in section 3.1.1. The implementation of a random strategy is key regarding the results validation. The results obtained through this strategy can be look at as a threshold to set in fact the utility of the remaining strategies.

The article [8], prepared by a group of authors of the faculty of Beijing, addresses popularity as an additional feature to recommender systems. The article begins by explaining the importance of the long tail[6] concept, stating that recommendations capable of retrieving objects from the end of this "tail" are much more valuable to the user. This feature largely depends on the dataset, since the balance between popular and unpopular items begins in the data collection. The authors use three datasets with different popularity distributions to demonstrate how distributions with longer tails result in recommendations with a greater bias towards popular items. The solution proposed in the article results in a Collaborative Filtering method based on Clusters. Each cluster will consist of items considered equally popular, making sure that the calculation of similarities is performed only between equivalently popular items. The work developed in this article springs as an inspiration to a set of strategies described in section 3.1.2.

The article [11] confirms the lack of tools able to mitigate the effect of popular items’ preponderance. To understand this effect the authors examine the relationship between the evaluator’s output characteristics - accuracy, popularity (as opposed to novelty) and diversity - with the characteristics of the user’s classification profile. There are two objectives of this analysis: (1) to probe the conditions under which common algorithms produce more or less diverse popular recommendations, and (2) to determine whether these custom recommendation algorithms reflect a user’s preference for diversity or novelty. The authors conclude that the diversity and popularity of the items present in the user’s profiles have little impact on the recommendations they receive. The article results in an interesting perspective that inspires part of the work developed in section 5.1, where we try to evaluate the created strategies created and assert if these result in more homogeneous recommendations regarding popularity.

3 FINDING ITEM NEIGHBORHOODS
First to implement the selected neighborhood strategies a tool for recommendations has been created. To ease the initial implementation effort we use Lenskit, an open source framework written in Java focused on the research environment, which several research projects have used before[1]. Based on Lenskit and its large collection of interfaces available we were able to create a modified version of this framework, available through open source in a GitHub repository[2]. The obtained results can be easily replicated using this tool and the annotated experimental configurations.

3.1 The Strategies
This study defines popularity as amount of times an item as been classified. Yet the distinction between sorting by the number of classifications, or by the mean value of these classifications is important. By using the number of ratings we ensure that the item was taken into account the largest (or lowest) number of times. In this popularity modeling quality is not taken into account, as often happens with the media content produced today. The intuition inherent to this idea rests on the human character of following the group dynamics, where the famous phrase “the rich gets richer” can be similarly defined as ‘the popular gets more popular’. Hence we take this in consideration and implement strategies that allow the study of both situations.

The implementation of these strategies follows the now exposed interface and the scheme of the figure 1, with special attention to the calculation of the respective set that each strategy returns. Interface NeighborIterationStrategy.java:

public interface NeighborIterationStrategy {
    LongIterator neighborIterator (...);
    void recompute (...);
}

3.1.1 Fixed Size Strategies: These set of strategies share a common template: every strategy starts by first ordering the total list of items and forming a queue. At each iteration of the cycle the next element of the queue is chosen and the similarity value is calculated through the chosen function. If the value is below the default threshold, the next item in queue is picked. This process holds until the neighborhood has N elements. All these strategies

[1]https://www.zotero.org/groups/lenskit/items/collectionKey/MQR6B47C
return a fixed set of $N$ elements, being $N$ injected as a parameter at the beginning of execution and it will be the same $N$ used to calculate predictions. Through this template four strategies are proposed:

- Sort by most popular items (those with higher amount of classifications)
- Sort by least popular items (those with smaller amount of classifications)
- Sort by best classifications (average value of items classification)
- Sort by worst classifications (average value of items classification)

An interesting point in these strategies is the ease in creating new strategies just by changing the sorting rule. For instance it would be possible to merge the number of classifications and the mean of classifications (using the geometric mean). Ordering through this measure we would instantly have a new strategy.

3.1.2 Adaptive Size Strategies: The previous strategies focused on building a queue through a heuristic of popularity and ensuring that items with higher priority are used more often as neighbors. These strategies don’t take into account the popularity of the item receiving the neighborhood itself. Adaptive strategies seek to explore this point providing neighborhoods that end up sharing the same degree of popularity with the items which receive them. However changing the content ain’t enough, it’s also necessary to adapt the neighborhood size around the popularity values. If we think about the less popular item situation we realize that its neighborhood would be composed only by unpopular items. It turns out that the probability of this construction being useful is in fact very low. Since these items are used only a few times, the number of users that this item shares with its neighborhood will always be very low. On the other hand the most popular set would have the opposite problem, counting on too many collisions that could end up as noise.

Two adaptive strategies have been implemented. First a strategy that follows an exponential decay function to determine the neighborhood size from the popularity value. And second a strategy that seeks, through the expected number of neighbors of the most popular item, to determine the fair number of neighbors for the remaining items. As adaptive size strategies both discard the recompute method. It doesn’t make sense to adulterate the neighborhoods with this method since the calculation of the neighborhood size is an intrinsic feature of these strategies.

- **Exponential Decay Strategy**

  For each item its number of occurrences is recorded and the neighborhood size $K$ is calculated through the expression 1. This expression has a set of constants that have a significant impact on the returned value. The decay constant ($d$) is responsible for the shape of the curve that dictates the size difference between popular and non-popular neighborhoods. The initial quantity constant ($a$) dictates the least popular neighborhood size, while the minimum value constant ($b$) ensures that the most popular neighborhood is composed by a minimal amount of elements. The value of each constant is chosen experimentally, this being the main limitation of this strategy

  \[ K = a e^{(-X \cdot d)} + b \]  

  (1)

  After $K$ is found it remains to find the first item in queue with the same popularity than the item in question and select the following $K$ items as neighborhood. Whenever there aren’t enough items in the remainder of the queue to satisfy the $K$, the selection reverses and fills the missing space with the immediately less popular items. Figure 2 depicts this process succinctly.

---

**Figure 2: Exponential Decay pseudo-code**

```python
1: Function decay_neighbor
2: Input: <i: itemId, a: int, b: int, d: int>
3: Output: <listNeighbor: array of itemIds>
4: Set empty listNeighbor
5: Reset queue
6: neighborSize = decay(i.popularity, a, b, d)
7: k = 0
8: While (k < neighborSize) or queue.hasNext() do
9:     neighbor = queue.next()
10:     if(neighbor == i)
11:         continue
12:     if(i.popularity <= neighbor.popularity)
13:         add neighbor to listNeighbor
14:     k++
15:     fill(listNeighbor, neighborSize - k)
16:     return listNeighbor
```
Analytically it’s possible to find $M(x)$ using $M(x)$, the function that describes an item’s neighborhood size given its popularity. To do so we just need to distribution $D(x)$ which describes the initial popularity distribution of the used dataset. $E(x)$ reveals how many entries there are for a given popularity value. Using $E(x)$ we can find how many entries there are for a given gap through $TotalE(x,n)$, and how many neighbors are for a given gap through $TotalV(x,n)$ like expressed in the expression 2.

$$E(x) = x \cdot D(x)$$

$$TotalE(x,n) = \sum_{y=x}^{n} E(y)dy$$

$$TotalV(x,n) = \sum_{y=x}^{n} D(y)dy$$

Thanks to the expression 2 we can now define $M(x)$ as the following expression 3. $I(condition)$ is the indicator function. We use it to confirm that $TotalE < N$ happens and sum the number of times this condition takes place. In the end the number of neighbors is simply the amount of times we have to sum a set of entries until $N$ is achieved times the number of neighbors in each moment.

$$M(x) = \sum_{n=1}^{Max} D(n) \cdot I((TotalE(x,n) < N))$$

In some scenarios $N$ is not achievable by just using the remaining items in queue. For instance the most popular item cannot search forward for the next more popular item. In these scenarios it must start looking backwards until $N$ is assured. The expression 4 describes the extra computation required every time $TotalE(x,Max)$ fails to secure $N$.

$$M(x) = TotalV(x,Max) + \sum_{n=x}^{Min} D(n) \cdot I((TotalE(x,n)+TotalE(x,Max)) < N)$$

This feature of the fair entries strategy allows the creation of strategies that follow in size an approximated distribution to $D(x)$. Unlike the exponential decay the fair entries strategy doesn’t require any constant setting and ends up following an inherent characteristic of the data.

4 EVALUATION

4.1 Datasets

Two different datasets are used: MovieLens 100k and MovieLens 1M. In order to understand their differences and their consequent impact on the results it’s essential to study the details of each one. In particular it’s crucial to capture how popularity differs. Both datasets are fruit of the work done under the GroupLens’ R&D project[7]. With time and more resources the remaining datasets from this group, bigger in size, shall be tested as well.

- **MovieLens 100k**: MovieLens 100k consists of 100,000 ratings (1-5) by 943 users over 1682 movies. Each user rated at least 20 films resulting in a density of 6.30%. Users who didn’t meet this requirement nor had their complete demographic information were removed. The information was collected through the MovieLens website during a period of seven months (19 September 1997 until 22 April 1998).

The figure 4 shows the distribution of popularity obtained from MovieLens 100k. From this plot it’s possible to see how a power-law function can describe the obtained distribution, and thus represent the observed decrease. Power-laws follow a set of well define properties like scale-invariance and universality. More than a hundred power-law distributions have been identified and studied in physics, biology, and the social sciences[4].

- **MovieLens 1M**: MovieLens 1M construction dates from the period between April 2000 and February 2003, resulting in 1,000,209 new classifications. During this period 6,040 users rated 3,760 films giving a density of 4.47%. The popularity distribution presented in figure 5 resembles to the distribution of the previous dataset. It maintains a power-law distribution with a slightly softer slope.

---

*https://movielens.org strengths and limitations of the proposed methodology.**
4.2 Methodology

**Lenskit** provides an evaluator however due the changes made to this framework (parallelization issues) we were forced to use Rival[^5], an open source tool developed in Java which allows a subtle control of the entire evaluation process[^5]. This way the evaluation of the different experiments is done by cross-validation with 5 folds splitting by user, based on three measures: RMSE, Precision and Recall and nDCG.

4.2.1 Experimental Configuration: *Lenskit* implements a specific system that assumes the responsibility of ingesting configurations. It is through this interface that the developer explores *Lenskit* as an experimental environment. Each configuration is written to a groovy extension file. All settings used in this project are found in the project’s [github](https://github.com/) repository along with the implemented solution.

- **Similarity function**: Cosine Similarity;
- **Minimal number of neighbors**: 2;
- **Threshold**: 0;
- **Centralization and Normalization**: Centralization applied to user vectors before calculating similarities. To each user rating is subtracted the user average rating;
- **Predictor**: Weighted average of similarities;

*Lenskit* default strategy is evaluated along with the created strategies providing a threshold to conclude whether there is progress regarding the current state of art or not. Two parameters related to the construction of neighborhoods are studied: the number of strategy neighbors and the number of predictor neighbors. The first number determines how many elements are used in calculating similarities and hence how many elements are available for calculating predictions. The number of predictor neighbors is always equal or less than the number of strategy neighbors. This number only indicates how many of the elements of the strategy neighborhood are actually considered in calculating predictions.

The created strategies were developed so that only depend on the number of strategies neighbors. Unlike fixed size strategies, adaptive strategies do not directly use this number. The exponential decay strategy which follows the expression \( I \) depends on a set of additional parameters that have been defined as follows:

- Initial quantity constant \((a) = \text{number of the most popular item’s entries}\)
- Minimum value constant \((b) = \text{number of strategy neighbors}\)
- Decay constant \((d) = 1/(a/b)\)

This is the main setback to any strategy that seeks to represent a parametric model. The fair entry strategy uses the number of strategy neighbors just to compute the total number of neighborhood entries available to the most popular item \((N)\). The *Lenskit* default strategy explores a method that makes use of cosine similarity’s symmetric and sparse characteristics in order to reduce the number of comparisons accomplished [10]. For a given item all users who have classified the item are traversed and all items classified by these users are collected. The selected number of strategy neighbors turns out to have no impact on this method, as confirmed in the next section.

5 RESULTS

5.1 MovieLens 100k Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Coverage</th>
<th>NDCG@10</th>
<th>RMSE</th>
<th>P@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most Popular</td>
<td>70%</td>
<td>0.583</td>
<td>1.044</td>
<td>0.579</td>
</tr>
<tr>
<td>Less Popular</td>
<td>1%</td>
<td>0.137</td>
<td>1.155</td>
<td>0.137</td>
</tr>
<tr>
<td>Best Classified</td>
<td>0.9%</td>
<td>0.137</td>
<td>1.204</td>
<td>0.139</td>
</tr>
<tr>
<td>Worst Classified</td>
<td>3%</td>
<td>0.148</td>
<td>1.225</td>
<td>0.170</td>
</tr>
<tr>
<td>Random</td>
<td>30%</td>
<td>0.406</td>
<td>1.113</td>
<td>0.514</td>
</tr>
<tr>
<td>Lenskit default</td>
<td>99%</td>
<td>0.801</td>
<td>0.938</td>
<td>0.756</td>
</tr>
<tr>
<td>Exponential Decay</td>
<td>81%</td>
<td>0.611</td>
<td>1.015</td>
<td>0.677</td>
</tr>
<tr>
<td>Fair Entries</td>
<td>92%</td>
<td>0.837</td>
<td>0.477</td>
<td>0.716</td>
</tr>
</tbody>
</table>

Table 1: MovieLens 100k. Strategy Neighbors = 20. Predictor Neighbors = 20

The first experiment noted in the table 1 immediately shows the superiority of Fair Entries strategy. Only *Lenskit* default approaches by means of coverage and P@10. The RMSE value obtained by Fair...
Entries is 2.14 times smaller than the average value of the experiment (1,021). The most accurate predictions by Fair Entries also suggest, through NDCG@10, that the 10 best ranked items per user end up better ordered than through Lenskit default. On the other hand P@10 suggests that the recommended list through Fair Entries ends up containing more irrelevant elements than the list created by the Lenskit default.

Fixed size strategies show difficulty at covering the evaluation set. Using 20 neighbors, only the Most Popular strategy covers more than 50% of the evaluation set. Any conclusion sought to be drawn through the remaining metrics becomes obsolete since there is no coverage to support them. The Most Popular method is limited to the 20 most popular items, i.e. with more entries. The fact that this method stands out from the other “siblings” portrays the impact in creating fortuitous environments that allow a greater number of collisions between items.

Table 2: MovieLens 100k. Strategy Neighbors = 100. Predictor Neighbors = 100

Increasing the number of neighbors through the experience in table 2 immediately improves the results from fixed size strategies. Coverage increases in all cases, but again only the Most Popular strategy covers more than 50% of the evaluation set. This strategy however achieves results very similar to Lenskit default. Interesting fact if we consider that only 100 elements are used and that the sorting of these elements only occurs strictly once during the whole process of similarity matrix construction. With about a hundred of the most popular elements it’s possible to achieve results as good or better than those created by the standard technique. Meanwhile the Random strategy is surprisingly reasonable. Without following any heuristic this method shows how, with the exception of the Most Popular, fixed size strategies are easily obfuscated by random neighborhood construction.

Table 3: MovieLens 100k. Strategy Neighbors = 100. Predictor Neighbors = 20

The experiment concerning the table 3 only changes the number of predictor neighbors regarding the previous experience. This experiment seeks to evaluate the impact in reducing the possible noise in a final prediction stage. The obtained values show an overall improvement, only getting worse at Lenskit default. Another couple of experiments were done to confirm this point. In all the configurations where the number of predictor neighbors were less than the strategy neighbors the results improved for the created strategies, suggesting that the predictor doesn’t need the less similar items previously computed.

Table 4: MovieLens 100k. Strategy Neighbors = 1400. Predictor Neighbors = 1400

In table 4 the number of strategy neighbors is aligned to a value close to the total number of items. The use of virtually all items as neighbors in fixed size strategies converges into lists that, although sorted differently, contain the same items regardless of strategy. The results thus converge in the same values. Fair Entries strategy follows this trend as well indicating that the constructed neighborhoods, required to contain a large share of all items, turn out to be similar to the neighborhoods from fixed size.

5.2 MovieLens 1M Results

Table 5: MovieLens 1M. Strategy Neighbors = 20. Predictor Neighbors = 20

The results obtained through the experiments with MovieLens 1M confirm many of the points described in the previous results section. With a greater amount of data the behavior of the strategies remains, except the Exponential Decay strategy. That said Fair Entries continues to outperform in the three metrics. Increased information intensifies this superiority. Using 100 strategy neighbors and 100 predictor neighbors it’s possible to cover 99% of the test set with very high accuracy (RMSE 0.367). Increasing the number of
neighbors improves the ordering and quantity of relevant items provided by Fair Entries but with a decreasing accuracy.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Coverage</th>
<th>NDCG@10</th>
<th>RMSE</th>
<th>P@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most Popular</td>
<td>96%</td>
<td>0.790</td>
<td>0.871</td>
<td>0.794</td>
</tr>
<tr>
<td>Less Popular</td>
<td>1.5%</td>
<td>0.142</td>
<td>1.071</td>
<td>0.165</td>
</tr>
<tr>
<td>Best Classified</td>
<td>3%</td>
<td>0.188</td>
<td>1.138</td>
<td>0.239</td>
</tr>
<tr>
<td>Worst Classified</td>
<td>6%</td>
<td>0.196</td>
<td>1.123</td>
<td>0.276</td>
</tr>
<tr>
<td>Random</td>
<td>72%</td>
<td>0.534</td>
<td>0.987</td>
<td>0.666</td>
</tr>
<tr>
<td>Lenskit default</td>
<td>99%</td>
<td>0.805</td>
<td>0.874</td>
<td>0.834</td>
</tr>
<tr>
<td>Exponential Decay</td>
<td>6%</td>
<td>0.245</td>
<td>1.112</td>
<td>0.229</td>
</tr>
<tr>
<td>Fair Entries</td>
<td>99%</td>
<td>0.962</td>
<td>0.367</td>
<td>0.844</td>
</tr>
</tbody>
</table>

Table 6: MovieLens 1M. Strategy Neighbors = 100. Predictor Neighbors = 100

5.3 Adaptive strategies showdown

What turns the Fair Entries strategy so imposing that at the same time makes the Exponential Decay so mediocre? The lone difference between these two strategies dwells in the number of elements selected for each neighborhood, all the remaining procedure is similar. The neighborhoods size by Exponential Decay follows an exponential distribution while the size of the Fair Entries neighborhoods is result of an entries sum in order to achieve the number of entries calculated from the most popular neighborhood.

The graph in figure 6 shows the size distribution of neighborhoods according to the popularity of the item. Distributions were captured during the experiment performed on table 1 in MovieLens 100k. Without surprises Exponential Decay strategy distribution follows the declared exponential expression while Fair Entries follows an approximated distribution of the initial popularity distribution as explained in section 3.1.2.

The difference between the two curves is immediately highlighted. The exponential curve moves closer to the minimum value at the same time as the “fair” curve offers a greater number of neighbors to items of average popularity. This difference is the main reason for the results mismatch between these two strategies.

In turn the good results obtained through Fair Entries might be explained by the strategy capacity of following the dataset’s popularity distribution. Figure 7 shows once again how Fair Entries follows up the dataset distribution now for MovieLens 1M. The power-law slope obtained is very similar to the one seen at section 4.1.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Coverage</th>
<th>NDCG@10</th>
<th>RMSE</th>
<th>P@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most Popular</td>
<td>99%</td>
<td>0.978</td>
<td>0.581</td>
<td>0.862</td>
</tr>
<tr>
<td>Less Popular</td>
<td>60%</td>
<td>0.451</td>
<td>0.971</td>
<td>0.357</td>
</tr>
<tr>
<td>Best Classified</td>
<td>88%</td>
<td>0.651</td>
<td>0.877</td>
<td>0.720</td>
</tr>
<tr>
<td>Worst Classified</td>
<td>99%</td>
<td>0.922</td>
<td>0.683</td>
<td>0.850</td>
</tr>
<tr>
<td>Random</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lenskit default</td>
<td>99%</td>
<td>0.805</td>
<td>0.875</td>
<td>0.834</td>
</tr>
<tr>
<td>Exponential Decay</td>
<td>75%</td>
<td>0.531</td>
<td>0.990</td>
<td>0.618</td>
</tr>
<tr>
<td>Fair Entries</td>
<td>99%</td>
<td>0.988</td>
<td>0.505</td>
<td>0.864</td>
</tr>
</tbody>
</table>

Table 7: MovieLens 1M. Strategy Neighbors = 1400. Predictor Neighbors = 1400
Figure 7: In red the function $M(x)$ which describes analytically the size of the neighborhoods obtained by Fair Entries. MovieLens 1M. Strategy neighbors = 100. Predictor neighbors = 100

6 POPULARITY EVALUATION

During the previous chapter it’s presented the metrics considered as state of art and the consecutive results according to these metrics. Through the obtained results it’s possible to draw some conclusions, such as which strategy is most accurate or which strategy provides the most relevant recommendations. However, in order to understand the impact of strategies on the levels of popularity, it is necessary to go beyond conventional methods of evaluation.

Table 8: Power-law parameters that best fit the number of recommended items distribution by strategy. In blue the values referring MovieLens 100k power-law. 100-100 experience

<table>
<thead>
<tr>
<th>MovieLens 100k dist</th>
<th>Scale</th>
<th>Slope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fair Entries</td>
<td>207,58</td>
<td>-1.109</td>
</tr>
<tr>
<td>Exponential Decay</td>
<td>271,07</td>
<td>-1.160</td>
</tr>
<tr>
<td>Lenskit default</td>
<td>13,13</td>
<td>-0.479</td>
</tr>
<tr>
<td>Random</td>
<td>221,75</td>
<td>-1.055</td>
</tr>
</tbody>
</table>

Table 9: Column 1 number of items recommended only once. Column 2 number of occurrences of the most recommended item. MovieLens 100k, 100-100 experience

<table>
<thead>
<tr>
<th></th>
<th>#once</th>
<th>#most</th>
<th>geo avg</th>
<th>geo avg norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lenskit default</td>
<td>76</td>
<td>361,56</td>
<td>0,263</td>
<td></td>
</tr>
<tr>
<td>Fair Entries</td>
<td>196</td>
<td>315,56</td>
<td>0,368</td>
<td></td>
</tr>
<tr>
<td>Exponential Decay</td>
<td>178</td>
<td>213,04</td>
<td>0,217</td>
<td></td>
</tr>
<tr>
<td>Most Popular</td>
<td>96</td>
<td>285,82</td>
<td>0,291</td>
<td></td>
</tr>
<tr>
<td>Less Popular</td>
<td>186</td>
<td>408,91</td>
<td>0,416</td>
<td></td>
</tr>
<tr>
<td>Best Classified</td>
<td>201</td>
<td>386,96</td>
<td>0,394</td>
<td></td>
</tr>
<tr>
<td>Worst Classified</td>
<td>149</td>
<td>328,67</td>
<td>0,334</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>176</td>
<td>342,36</td>
<td>0,348</td>
<td></td>
</tr>
</tbody>
</table>

With this said we use the geometric average between the number of items recommended once and the occurrence number of the most recommended item. An ideal distribution intersects it’s axes as early as possible. The geometric mean of these two numbers gives an interesting perspective of this point. The idea is to compress these two numbers in a single metric and search for the lowest average. For instance since we give equal weight to both measures, Lenskit default largely benefits from a low number of items only recommended once, but it ain’t enough to beat the low Exponential Decay’s occurrences of the most recommended item. Even though it describes correctly the harmony between the distributions edges we still miss the understanding of the distributions behavior that an approximated slope would offer. Table 9 shows the geometric average of the first two columns as the geometric average from normalized columns.

Another accessible way to find out the popularity balance of a strategy is to understand how easily it recommends an item. To quantify this feature we calculate the cumulative probability distribution of recommending an item at least Y times. Figure 8 shows the respective probabilities for all strategies.

First of all, no strategy ensures that it recommends at least once for all items. The certainty ranges between 10% and 60% through the Less Popular and Random strategies respectively.

Secondly the probability of recommending as many times as possible an item is very low, about 0.06%. The ideal cumulative probability distribution intersects Y as early as possible while providing a 100% probability of recommending at least once any item.

This probabilistic technique allows us to evaluate two important points in the long tail problem: how easily each system recommends an item and the probability in recommending many times. These points may result in a metric that attempts to compare the probability of recommending an item at least N-times - Probability@$N$. For instance the Fair Entries’ probability@6 is about 28% while the default strategy is about 11%. Fair Entries offers better guarantees at reaching long tail items. The Random strategy is surprisingly effective in this assessment. However, one must be aware of the inflection point. At certain $N$ the probability@$N$ meaning inverts as we seek smaller probabilities for $N$ too large. The inflection point depends on the balance sought and the distribution itself.
During the previous evaluation, 20 recommendations were required for each of the 943 MovieLens 100k users. The required number is at the application discretion and is often neglected during the evaluation process. However, this number has a direct impact on the cumulative probability.

Figure 9 presents the cumulative probability distributions of the Fair Entries strategies in different scenarios. By increasing the number of recommendations we naturally increase the guarantees of recommending an item at least once. Likewise, the maximum number of times an item can be recommended increases. It’s not very clear to figure out which number to use. In the end it comes down to a compromise between securing recommendations in exchange of more popular items in our recommendations.

The context in which these recommendations take place makes a big difference as well. In this technique the recommendation ranking is not taken into account. In Top-N problems whenever we ask for more recommendations we can be sure that these are classified as “worse” than the first batch. From 10 to 40 we guarantee that we recommend more items but if the 30 additional recommendations are not used these guarantees are of little use. For this reason, either the number of recommendations requested is in line with Top-N or the problem runs out of the Top-N context.

7 CONCLUSIONS

Through this study we draw strong evidence that popularity as an item’s feature expresses an important component for the recommendations stage. In this section we will summarize the conclusions drawn throughout the article and highlight some important points.

The most popular set of items is efficient at building item neighborhoods.

Although the Most Popular strategy doesn’t provide the best set of results, its prominence regarding other strategies is surprising. In experiments where the number of selected neighbors is small the use of the most popular items allows a reasonable coverage. With about a hundred of the most popular items we can already achieve similar results to Lenskit default. This means that in order to get satisfactory recommendations it’s only necessary to perform a hundred comparisons per item. The Fair Entries and Lenskit default strategies use a higher average number of neighbors, and besides, they need to apply the neighborhood heuristics to each item. Most Popular strategy applies its heuristic only once, in ordering a queue available to all items. The efficiency of the Most Popular strategy becomes evident.
The content of the neighborhood is accountable for an increased precision of long tail items’ predictions.

Adaptive size strategies ensure that similarities used at the prediction moment are associated exclusively to items of identical popularity. This is a very important feature. By adapting the neighborhood of an item we ensure that users who rate a large amount of items will not mislead the predictor, since the predictor only searches for affinities among items of the same popularity. On the other hand, users who rate unpopular items will have adequate predictions for less popular items, something that fixed-size strategies such as Most Popular fail to do. Most Popular strategy just establishes that the predicted classification for an item properly expresses its affinity with the most popular items (assuming that the user has rated the most popular items). Although useful this prediction fails to detect the actual user interest regarding the less popular item.

We expect that whenever an user ranks a less popular item \( i_1 \) that rank is used to predict the user’s taste for another less popular item \( i_2 \). It turns out that the Most Popular strategy ignores \( i_1 \), reporting only the affinity of \( i_2 \) with the most popular ones. Fair Entries takes a different approach and seeks to detect all the user’s ranked items with popularity similar to \( i_2 \), thereby predicting \( i_2 \) based on \( i_1 \).

If users actually rate the most popular items, it’s reasonable to predict based on the users’ interest for these items, like we discussed in the previous point. However the long tail problem reports that the vast majority of items are in the non-popular spectrum, and within that spectrum we find a diverse amount of niches. In order to find these niches we have to move away from the generalization applied by strategies such as Most Popular and base our predictions on items of identical popularity. The results obtained through the strategy Fair Entries support this practice. The astonishing low RMSE implies a deep level of detail, almost as if we were overfitting but without losing the predictive power for unseen user-item pairs. To expose the long tail niches this detail is crucial and obtainable through neighbors of adaptive content.

The size of the neighborhood has direct impact on the recommendations. Strategies sensitive to the dataset popularity distribution result in better recommendations.

The previous point refers to the importance of using items of identical popularity at the prediction moment. To ensure that the similarities between these items are used in the prediction process it’s necessary to adapt the amount of these similarities, by the size of the neighborhoods, to the popularity under the light.

In section 5.3 a Fair Entries feature is presented as justification of its superiority against Exponential Decay. The size of its neighborhoods follows a distribution similar to the dataset’s inverse distribution of the amount of items by popularity. That is, the size of the neighborhood is inversely proportional to the popularity of the item, a proportion that is automatically prescribed by the characteristics of the dataset. By taking special consideration in how an item’s popularity fits the overall popularity spectrum, the strategy is sensitive to the context and provides a suitable neighborhood in size. This feature and the feature set forth in the previous

point are together the beacon that highlights Fair Entries. This strategy doesn’t just result in the best-valued strategy according to the metrics noted in chapter 5, but it also shows an increased facility at recommending all kinds of items while maintaining a balanced proportion of popular and unpopular items, as seen in chapter 6.

Using less information in the predictor has positive consequences.

While retrieving results several experiments were executed where the number of predictor neighbors was tested. We sought to understand the impact of this number and its relation with the number of strategy neighbors. We know that this number is limited by the neighbors provided by the strategy, as the set of predictor neighbors is contained in the set of strategy neighbors. By reducing the number of predictor neighbors regarding the number of strategy neighbors the results are considerably better. Except for the default strategy, all strategies benefit when reducing the amount of predictor neighbors. The predictor, upon receiving the set of an item’s neighbors, uses the intersection of the item’s neighborhood and the user’s associated items. The experiments suggest that using the less similar items of this intersection increases the undesirable noise, so by decreasing the number of predictor neighbors the less similar items are excluded thus improving the prediction and the final results consequently.

REFERENCES