Explicit Elimination of Similarity Blockers for Session-based Recommendation

Mattia Brusamento[1], Roberto Pagano[1], Martha Larson[2], and Paolo Cremonesi[1]

[1]DEIB, Politecnico di Milano, Milan, Italy, <name.surname>@polimi.it

[2] Delft University of Technology, Delft & Radboud University, Nijmegen, Netherlands, m.a.larson@tudelft.nl

ABSTRACT

A single 'odd' interaction can cause two user interaction sessions to diverge in similarity, and stand in the way of generalization. The sensitivity of session-based recommenders to session similarity motivates us to explicitly identify and remove such 'similarity blockers'. Specifically, we leverage huge amounts of data, which allow us to identify blockers in the form of non-co-occurring items. Other blockers can be identified using content-based similarity. Our experiments reveal that explicitly eliminating relatively few blockers improves performance.

Keywords

Near-duplicates; Session-based; Large-scale; *30Music* dataset

1. INTRODUCTION

Session-based recommender systems leverage information about the current user session to predict how it will continue. Session-based methods often rely heavily on similarity between sessions. A single aberrant user interaction in a session has the power to cause two otherwise highly similar sessions to suddenly become dissimilar. We refer to such an interaction as a 'session blocker', since it causes two sessions to be different without reflecting a real underlying difference of user preference.

Session blockers can arise in two ways. First, a user may interacts with an item only once, incidentally and unrelated to preference. Second, the user may click on a preference-related item, but it is a near-duplicate, and has an unexpected item ID.

The current era of big data opens the ability to search for individual interactions that may have been incidental clicks. In the past, non-occurrence, or non-co-occurrence of items could be attributed to lack of data. Now, however, it makes sense to investigate whether this non-co-occurrence is truly incidental or whether it could be considered the source of additional information that can be exploited for further improve recommendations. For completeness we also look at conventional near-duplicates identified on the basis of their metadata. We are inspired by our previous work on near-duplicates in large collections [3].

In this paper, we show that considering items similar due to non-co-occurrence, and, more conventionally, on the basis of metadata will improve recommendation effectiveness. The effect may be subtle, but a larger implication is that we should not assume that items occurring in similar sessions contribute positively to recommendations.

RecSys 2016 Poster Proceedings, September 15-19, 2016, Boston, USA .

2. REDUNDANCY AND RECSYS

We identify items (pairs or small clusters) that have different item IDs, but should be treated as the same item, since they will otherwise introduce a non-negligible 'similarity blocking' effect. We demonstrate two points: (1) these items should not be recommended together (since they are redundant) and (2) collapsing them before training a model could enhance performance (by eliminating 'blocking').

We adopted two different approaches to find similarity-blocker items. The first and more straightforward approach, called *technical duplicates*, is based on the idea of detecting near-duplicates in the collection of songs. In our case, the available metadata is a Last.fm[1] URL, consisting of a string containing artist and title. We apply some basic NLP techniques to compute the similarity between two items: after properly parsing the URL, we compute the Jaccard similarity among the resulting words. Since we are looking for near-duplicates, we apply a threshold to decide which items should be considered equal.

The second and more sophisticated approach, called *collaborative duplicates*, leverages the fact that two items fail to ever co-occur despite their occurrence in similar (but separate) contexts. The idea finds support in our observation that, for a given user, the similarity between sessions containing *technical duplicates* is significantly higher than the average pairwise session similarity: this suggests that we can expect near-duplicates to appear in similar sessions from the same user. Moreover, we expect two near-duplicates never to co-occur in the same session, based on the rationale that the user would not listen to both of them together.

Next, we describe our technique. During training, we consider the set of sessions from each user in turn, and compute the pairwise Jaccard similarity on this set. We retain the pairs above a certain threshold. From these pairs of sessions, we extract pairs of items, such that one belongs to one session and one to the other, with the constraint that they must not co-occur in a session the training set for any user. The score given to the pairs of items is calculated as the average similarity between the pairs of similar sessions from which they come from, with a *shrinkage factor*. Finally, we keep the pairs of items with a score above the average, thus creating a cluster for each of these items. In

[1]http://www.last.fm

this way, many cluster overlap each other, and creating the transitive closure of the clusters will result in a unique single big cluster containing all the clustered songs. Since we are instead interested in micro-clusters we kept only the disjoint clusters, which usually contain 2 or 3 items.

3. DATA AND EXPERIMENTS

We carried out our experiments with an already existing large scale music recommender system: the *implicit playlist recommender (IPR)* [1]. The algorithm is trained using the user listening sessions: similar sessions from the same user are considered as *implicit playlists*. The recommendations are performed by matching the current session against the *implicit playlists*, the songs from the best matching playlists are then recommended. Our goal is to improve the performance of this algorithm by conflating redundant items. The data we used come from the *30Music* dataset [2], thus the size of the catalog is quite big (about 4M songs). We adopted Apache Spark to deal with the large-scale of the data. In particular, we used a cluster composed by 200 nodes.

To compute technical duplicates, we applied some heuristic optimization, such as the assumption that two near-duplicates should have at least one bigram in common. In this way, we avoid comparing each item with all the others. By leveraging the map-reduce paradigm, our algorithm took about 4 hours to identify near-duplicates. As one could expect, using this technique we find redundant items as: 'Chris James feat. Ria Moran - Song For Her' and 'Chris James - Song for Her (feat. Ria Moran)'. In this way, we reduced the number of items by 6.09%.

As for the *collaborative duplicates*, the number of items we were able to find was much lower (around 600). All of them belong to the long tail, but come from the most active users, who have a higher probability of contributing.

Table 1: Collaborative Duplicates Examples

Brutal Truth - Swift and Violent (Swift Version) Brutal Truth - Swift and Violent - Swift Version
Luis Miguel - La Incondicional Luis Miguel - Cuando,vuelva a tu lado
Underoath - Moving for the Sake of Motion Demon Hunter - Less Than Nothing

Table 1 contains three characteristic examples of *collaborative duplicates*: the first illustrates two songs that are technically equivalent and are detected also by the first technique. The second example shows two songs from the same artist. Note we do not claim that two songs from the same artist should be automatically considered redundant. The last example is the hardest to classify: these two songs have a similar content, in the sense that they have the same acoustic features. In this case, our mining technique shows the ability to spot also pairs of songs that are not straightforward to classify even by a human.

4. RESULTS AND DISCUSSION

We adopted the original experimental settings of the IPR[2] used in [1]. We ensure score comparability by projecting the recommendations generated by IPR into the cluster space of the experimental conditions. Note that this process advantages the baseline, meaning that smaller differences in score can be considered more important. Figure 1 shows the differences between the performance obtained by applying the clustering and the original IPR. We see that applying the combination of the *collaborative* and *technical* redundancy detection (Coll&J75 and Coll&J9), leads to a positive difference in the performance for all N, and has positive effects in the training of the model. Precision, omitted here for space reasons, shows similar behavior.

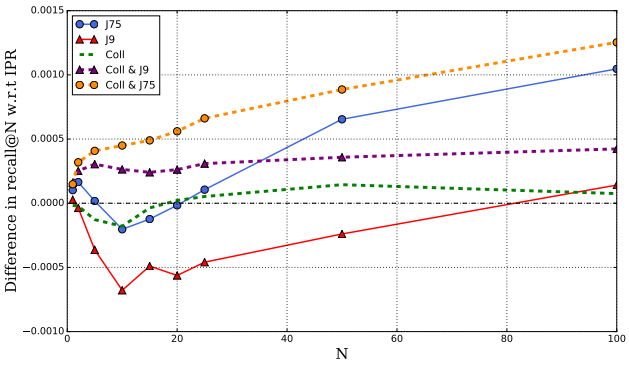

Figure 1: Impact of clustering on recommendation

The difference in performance can be explained by the fact that merging redundant items allows the creation of more playlists. In all the cases, our techniques improve the performance for big N. This is because the original algorithm poses an hard threshold on a soft similarity (shrunk Jaccard) and, by merging duplicates, a larger number of useful *implicit playlists* are above the similarity threshold. In the future, we plan to analyze the effects with different recommendation algorithms and also to evaluate online.

Finally, we double-checked our assumptions by mapping our new recommendations into the original space, which we did by flattening the clusters. As one would expect, this degrades the performance. This result confirms our conclusion that the groups of items we identified as redundant should not be recommended together.

The findings in this paper are interesting in light of the currently growing importance of session-based recommendation. Effectively, we have shown that not all similarity is good: instead, items can potentially be *too* similar. The approach of eliminating such similarity-blocker items explicitly, is greater than might be otherwise assumed, given their relative limited numbers. Merging items before training a model can boost performance, and, moving forward, may also have important implications for diversity: by removing too-similar items, more slots open for other items.

5. REFERENCES

[1] R. Turrin, A. Condorelli, P. Cremonesi, R. Pagano, and M. Quadrana. Large scale music recommendation. *Workshop on Large-Scale Recommender Systems (LSRS 2015) at ACM RecSys*, 2015.

[2] R. Turrin, M. Quadrana, A. Condorelli, R. Pagano, and P. Cremonesi. 30Music listening and playlists dataset. *ACM RecSys poster 2015 CEUR-WS.org/Vol-1441/recsys2015_poster13.pdf.*

[3] R. Vliegendhart, M. Larson, and J. A. Pouwelse. Discovering user perceptions of semantic similarity in near-duplicate multimedia files. *CrowdSearch 2012 CEUR-WS.org/Vol-842/crowdsearch-vliegendhart.pdf.*

[2]https://github.com/mquad/playlistrec