Cluster-based Contextual Recommendations

Kostas Stefanidis
ICS-FORTH, Greece
kstef@ics.forth.gr

Eirini Ntoutsi
LMU, Germany
ntoutsi@dbs.ifi.lmu.de

ABSTRACT

In this work, we address the problem of contextual recommendations by exploiting the concept of fault-tolerant subspace clustering. Specifically, we pre-partition users that have similarly rated subsets of data items into clusters and associate with each cluster a context situation. Context is defined as any internally stored information that can be used to characterize the data per se. Then, given a query context, we identify the clusters with the most similar context, and use their members for making suggestions in a collaborative filtering manner.

1. DESCRIPTION

Recommender systems have become indispensable for several Web sites, such as Amazon, Netflix and Google News, helping users to navigate through the infinite number of available choices. Motivated by the fact that often users have different preferences under different context situations, several approaches, e.g., [1], extend recommender systems beyond the two dimensions of users and items to include further contextual information. Context can be defined as any *external* to the database information that can be used to characterize the situation of a user, such as the location, time or companion of the user, or any internally stored information that can be used to characterize the data per se [6]. In our work, we follow an internal contextualization approach, and infer context from data. A simple way to express an internal context is by specifying conditions for the presence of particular attribute values in the data. For example, for a movies recommender, an internal context can be: genre=comedy & production-year=2015. It is clear that such a context characterization cannot be done upon the whole database, as the data display a lot of variability. Rather, we should look for contextual information in smaller, homogenous subgroups of our data.

To extract contextual information, we rely on similarities on the user ratings. Intuitively, users close together in their ratings, share the same context let it be the preference for similar movies type, or preference towards a specific director. Typically the user similarity is evaluated w.r.t. the full dimensional feature space, i.e., all available items. Finding users similar for all different items though is hard, while it is more reasonable to find users similar w.r.t. a subset of the items. A straightforward approach to derive such subsets is to categorize the items based on some domain knowledge. In case of movies, for example, the movie genre can be used and items that belong to the same genre can form a subset. A problem with this approach is that such categories are quite vaguely defined, diverse and also overlapping. For instance, the movies *Ted* and *My big fat Greek wedding* are both classified under *comedy*, the later however can be also found under *romance*. For a user interested in comedies it is not clear whether he/she would equally appreciate a suggestion on *Ted* and *My big fat Greek wedding*. Such a general items categorization, does not reveal much about the aspects that bring users together. Moreover, such aspects might be beyond some given categorization, like the movie genhre. Ideally, we want to find subsets of items which are rated similarly by some users; such a subset implies that there is something in common in these items that places these users together. This does not need to be that generic as the genre, but it might be some other common property of the items, like the director, the story, or even a mixture of them.

In [4], we locate such user-item groups by exploiting (fault-tolerant) subspace clustering. Subspace clustering is a popular approach for clustering high dimensional data which discovers, except for the cluster members, the dimensions upon which these members form a cluster. Different subspace clusters might be defined upon different subspaces and member overlaps in the clusters are allowed. In our case, subspace clustering identifies groups of users with similar behavior w.r.t. a set of items. We employ the items of a subspace cluster to build its context and use it to locate, at query time, clusters with context similar to the query context. In contrast to our prior work [4] that considers all user-related clusters for issuing recommendations, here we consider only context-related clusters for the specific user.

Recommendations Basics: Assume a recommender system, where I is the set of items and $\mathcal{U}$ is the set of users. Each item $i \in I$ is described as a set of (attribute, value) pairs; let $\mathcal{D}$ be the set of all distinct (attribute, value) pairs appearing in all data items. For instance, for a movies application, an attribute can be the director or the production year of a movie. A user u might rate an item i with a score $rating(u, i)$ in $[0.0, 1.0]$; let R be the set of all ratings recorded in the system. Typically, the cardinality of I is high and users rate only a few items. For an i, unrated by u, with N_u representing u's most similar users (neighbors), its relevance score is computed as:

$$relevance(u, i) = \frac{\sum_{u' \in N_u} simU(u, u') rating(u', i)}{\sum_{u' \in N_u} simU(u, u')} \quad (1)$$

where the similarity function $simU(u, u')$ evaluates the proximity between u and u'. The most prominent items, i.e., those with the higher relevance, are suggested to the user.

Fault-tolerant Subspace Clustering: Subspace clustering aims at detecting clusters embedded in subspaces of a high dimensional dataset. Clusters may consist of different combinations of dimensions, while the number of relevant dimensions per cluster may vary strongly. To restrict the search space, only axis-parallel subspaces are searched though for clusters. A *subspace* S describes a subset of items, $S \subseteq I$. A subspace cluster C is then described in terms of both its members $U \subseteq \mathcal{U}$ and subspace of dimensions $S \subseteq I$ upon which it is defined as $C = (U, S)$. Typically, subspace clustering does not deal with missing values, which is a key problem for recommendations. Fault tolerant subspace clustering [3] deals with this issue by allowing a certain amount of missing values per items, users and ratings in a cluster.

In [4], we use fault tolerant subspace clustering to locate users with similar preferences to a query user, for computing his recommendations. In particular, for a query user u we locate its similar users via the subspace clusters where the user belongs to. These are locally similar users, the term "locally" meaning that they are similar w.r.t. a set of dimensions (those in their corresponding subcluster). We refine this set of users based on their common ratings to u; this is a "global" evaluation aiming to check their overall proximity, i.e., over all items. This local-global refinement results in a more qualitative set of friends N_u for recommendations. The new set N_u is plugged in Formula 1 for issuing recommendations. Our results show that this careful selection of friends, is reflected in quality of recomemndations.

Inferring the Cluster Context: For each subspace cluster, extracted upon our ratings dataset, we infer its context. In particular, we consider that the context of a subspace cluster $C = (U, S)$ expresses the most significant parts of the items S within the cluster; these are captured through the attribute values of the items of the subspace S, upon which C is defined and are therefore sets of (attribute, value) pairs. Similar to [5], we ground the significance of each (attribute, value) pair on its frequency in the data appearing in the cluster. By post-processing the (attribute,value) pairs in S, we rank these pairs based on their frequency in C; the significance of a pair is normalized taking into account its frequency in the whole database, so as to downgrade global popular pairs and focus on cluster-specific context. This way, we define the context of a cluster as an expression containing one or more significant (attribute,value) pairs. For instance, the context of a movie cluster could be: genre=comedy & actor=Meryl Streep.

Luckily, our subspace clustering clustering is an offline procedure, meaning that there is no need to compute at query time the contexts of the produced clusters. This fact allows us to resorting to non-approximate solutions for context identification.

Contextual Recommendations: Given a user u along with a query with context p, expressed as a set of (attribute, value) pairs with attributes coming from $\mathcal{D}$, for computing contextual recommendations for u, we first locate the users that exhibit the most similar behavior to u under p. These are the members of the clusters for which u is also a member; we denote them by C_u. Due to the context-constraints though, not all clusters are relevant as some of them describe a different context than p. Therefore, we need a way to evaluate the relevance of a cluster context to p. We distinguish between:

•*Exact context match*: If there are clusters in C_u that match exactly the query context p of u, i.e., C_u^p, then the members of these clusters comprise the set of friends N_u upon which the recommendations for u will be computed.

•*Partial context match*: If there is no cluster with context equal to p, we relax our context relevance evaluation by looking for context-similar clusters, instead of context-identical clusters. To determine how close a context query p and a cluster context c are, we rely on a vector-based approach. Let $\mathcal{D}$ be the set of all N distinct (attribute, value) pairs appearing in all data items. A vector representation of p is a binary vector V_p of size N, whose j-th element corresponds to $\mathcal{D}[j]$. If $\mathcal{D}[j]$ appears in p, then $V_p[j] = 1$; otherwise it is 0. Analogously, the vector representation of a cluster context w is a binary vector V_w of size N, where $V_w[j] = 1$, if $\mathcal{D}[j]$ appears in w; otherwise it is 0. The similarity between p and w is then defined using their vector representations V_p and V_w as:

$$sim(p, w) = cos(V_p, V_w) = \frac{V_p \cdot V_w}{|V_p||V_w|} \quad (2)$$

Having located the clusters C_u^p with the most similar contexts to p, we employ their members as the set of the most similar users to u and compute recommendations based on them. Actually, we apply a weighted ranking approach to refine the set of like-mined users according to the similarity of the context of the cluster they belong to, to p.

Next Steps: We are working on improving our cluster context description, by a better aggregation of the attribute values within the cluster and by using item hierarchies, and on more sophisticated methods for context matching and user aggregation. Also, we are working on the scalability aspect to parallelize the subspace cluster and context extraction. Some preliminary results with MapReduce appear in [2]).

2. REFERENCES

[1] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, 2005.

[2] V. Efthymiou, K. Stefanidis, and E. Ntoutsi. Top-k computations in MapReduce: A case study on recommendations. In *IEEE Big Data*, 2015.

[3] S. Günnemann, E. Müller, S. Raubach, and T. Seidl. Flexible fault tolerant subspace clustering for data with missing values. In *ICDM*, pages 231–240, 2011.

[4] E. Ntoutsi, K. Stefanidis, K. Rausch, and H. Kriegel. Strength lies in differences: Diversifying friends for recommendations through subspace clustering. In *CIKM*, 2014.

[5] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *SIGMOD*, 1996.

[6] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19, 2011.