Exploring Subspace Clustering for Recommendations

Katharina Rausch
LMU, Munich
rausch.katharina@gmail.com

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

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

Hans-Peter Kriegel
LMU, Munich
kriegel@dbs.ifi.lmu.de

ABSTRACT

Typically, recommendations are computed by considering users similar to the user in question. However, scanning the whole database of users for locating similar users is expensive. Existing approaches build user profiles by employing full-dimensional clustering to find sets of similar users. As the datasets we deal with are high-dimensional and incomplete, full-dimensional clustering is not the best option. To this end, we explore the fault tolerance subspace clustering approach that detects clusters of similar users in subspaces of the original feature space and also allows for missing values. Our experiments on real movie datasets show that the diversification of the similar users through subspace clustering results in better recommendations comparing to traditional collaborative filtering and full dimensional clustering approaches.

1. INTRODUCTION

With the growing complexity of WWW, users often find themselves overwhelmed by the mass of available choices. Shopping for DVDs, books or clothes online becomes more and more difficult, as the variety of offers increases rapidly and gets unmanageable. To facilitate users in their selection process, recommendation systems provide suggestions on items, which could be interesting for the respective user. Typically, user recommendations are established by considering users with similar preferences to the query user. Scanning the whole database to find such like-minded users, though, is costly. More efficient approaches have been proposed that build user models and exploit these models for recommendations. For example, [4] applies full-dimensional clustering to organize users into clusters and uses these clusters, instead of linear scanning the database, for predictions.

Full dimensional clustering though, is not the best option for the recommendation domain, where there might exist thousands to millions of items and therefore, finding similar users with respect to all the dimensions might be difficult. Feature reduction techniques like PCA are not appropriate for this problem since they provide a global reduction of the feature space, which is not appropriate for cases where different dimensions are relevant for different clusters. For example, there might exist a cluster of users based on comedies, part of which might belong to another cluster based on dramas, but there might be no cluster of users with interest in both comedies and dramas. To deal with this issue, we propose to employ *subspace clustering* [3], that extracts both clusters of users and dimensions, i.e., items, based on which users are grouped together. There is also another motivation behind the subspace clustering choice: as users possibly belong to more than one subspace cluster (each cluster defined upon different items), this broadens our options for selecting like-minded users for a given user. Moreover, employing in the recommendation process users that differ qualitatively in terms of the items upon which their selection was made, diversifies the set of like-minded users.

The rest of the paper is organized as follows. Section 2 offers an introduction on recommendations. Section 3 overviews fault tolerance subspace clustering and explains how it can be employed for recommendations. Section 4 presents our experimental results and Section 5 concludes our work.

2. BACKGROUND AND LIMITATIONS

Assume a recommendation system, where I is the set of items to be rated and U is the set of users in the system. A user $u \in U$ might rate an item $i \in I$ with a score $rating(u, i)$ in $[0.0, 1.0]$; let R be the set of all ratings. Typically, the cardinality of the item set I is high and users rate only a few items. The subset of users that rated an item $i \in I$ is denoted by $U(i)$, whereas the subset of items rated by a user $u \in U$ is denoted by $I(u)$.

For items unrated by users, recommendation systems estimate a relevance score, denoted as $relevance(u, i)$, $u \in U$, $i \in I$. In this work, we follow the collaborative filtering approach: similar users are located via a *similarity function* $simU(u, u')$ that evaluates the proximity between $u, u' \in U$. We use F_u to denote the set of the most similar users to u, hereafter, referred to as the *friends* of u: $\mathcal{F}_u = \{u' \in \mathcal{U} : simU(u, u') \geq \delta\}$, where δ is a similarity threshold.

Given a user u and his friends $\mathcal{F}_u$, if u has expressed no preference for an item i, the relevance of i for u is given by:

$$relevance_{\mathcal{F}_u}(u, i) = \frac{\sum_{u' \in \mathcal{F}_u} simU(u, u') rating(u', i)}{\sum_{u' \in \mathcal{F}_u} simU(u, u')}$$

After estimating the relevance scores of all unrated user items, the top-k rated items are recommended to the user.

One of the key issues in collaborative filtering approaches

is the identification of the set $\mathcal{F}_u$ of a user $u \in U$. The two general strategies in this direction are: (i) *Naive approach*: Retrieve all users U and return as friends of a user u, those whose similarity to u is above the threshold δ. (ii) *Full dimensional clustering approach*: Partition users in U based on their similarity, so that, users that exhibit high similarity are placed in the same cluster. The friends of a user are all those users belonging to his cluster. The typically considered clustering here, is full dimensional, i.e., over all items of I.

The naive approach would be extremely inefficient in large systems, since it requires the online computation of the set of friends for each query user. Moreover, since this appears to be an online method, the set of friends should be computed from scratch every time a user request arrives, even if the user has requested recommendations just before. One way to overcome the limitations of the naive approach, is to build some users model and directly employ this model for recommendations. Full dimensional clustering has been used towards this direction to organize users into clusters of similar ones [4]. The pre-computed clusters are then employed to speed up the recommendation process; the friends of a given user u correspond roughly to the users that belong to the same cluster as u. However, not all similar users to u belong to its cluster and therefore the friends set might be incomplete and too narrow.

3. SUBSPACE CLUSTERING FOR USER RECOMMENDATIONS COMPUTATION

Subspace clustering aims at detecting clusters embedded in subspaces of a high-dimensional dataset [3]. 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 through 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 $C = (U, S)$.

The vast majority of subspace clustering algorithms works on complete datasets. However, our data is characterized by many missing values, since users rate only a few items. Recently, *fault tolerant subspace clustering* [2] has been proposed to handle sparse datasets. The main idea of this approach is that clusters including missing values can still be valid, as long as the amount of missing values does not have a negative influence on the cluster's grade of distinction.

To restrict the number of missing values in a subspace cluster, thresholds with respect to the number of missing items, the number of missing users and their combination, are used. These thresholds are adapted from [2] to the recommendations domain. Users featuring a missing value for item $i \in I$ are included in $U_?(i) = \{u \in U | rating(u, i) =?\}$, whereas $I_?(u) = \{i \in I | rating(u, i) =?\}$ holds those items having a missing value for user $u \in U$.

User Tolerance: Each user in a subspace cluster must not contain more than a specific number of missing item ratings. That is, $\forall u \in U : |I \cap I_?(u)| \leq \epsilon_u \cdot |I|$, where $\epsilon_u \in [0, 1]$ is the user tolerance threshold.

Item Tolerance: Each item in a subspace cluster should not contain too many missing values. That is, $\forall i \in I : |U \cap U_?(i)| \leq \epsilon_i \cdot |U|$, where $\epsilon_i \in [0, 1]$ is the item tolerance threshold.

Pattern Tolerance: The total number of missing values in a subspace cluster must not exceed the *pattern tolerance* threshold $\epsilon_g \in [0, 1]$. That is, $\sum_{u \in U} |S \cap I_?(u)| \leq \epsilon_g \cdot |U| \cdot |S|$.

Thus, a cluster $C = (U, S)$ is a *valid fault tolerant subspace cluster* if the number of missing items per user does not violate ϵ_u, the number of missing users per item does not violates ϵ_i and the total number of missing values is bounded with respect to ϵ_g.

Bottom-up subspace clustering approaches make use of the *monotonicity property*: if $C = (U, S)$ is a subspace cluster, then in each subset S', $S' \subseteq S$, there exists a superset of users, so that, this set is a subspace cluster as well. The fault tolerance model defined so far, does not follow the monotonicity property. Therefore, [2] suggests *enclosing cluster approximations*, which form supersets of the actual clusters, i.e., they include more users than the actual subspace clusters do. These approximations follow the monotonicity property. A subspace cluster can be extended by adding some dimensions up to a dimensionality of mx. Thus, each fault tolerant subspace cluster $C = (U, S)$, with $|S| \leq mx$, is *mx-approximated* by a maximal fault tolerant subspace cluster $A = (U_A, S)$, established by the thresholds $\epsilon_u = min\{\epsilon_u \cdot \frac{mx}{|S|}, 1\}$ and $\epsilon_i = \epsilon_g = 1$. The rationale is to extend the subspace clusters by some dimensions, in order to create enclosing approximations, which fulfill the fault tolerance thresholds.

In [2] the *grid-based fault tolerant subspace clustering* algorithm $FTSC$ is proposed that integrates the fault tolerance concepts to the grid-based subspace clustering algorithm CLIQUE [1]. As in CLIQUE, the data space is partitioned into non-overlapping rectangular grid cells by partitioning each item into g equal-length intervals and intersecting the intervals. Clusters consists of dense cells, where density is quantified in terms of a density threshold $minPts$.

In $FTSC$, users with missing values/ratings might also be part of the clusters. To this end, an extra interval is allocated for each item, where users with missing values for the respective item are placed in. To generate cluster approximations, except for the item intervals with existing values, we also consider the item intervals for missing values. Thus, a cluster approximation consists of the users in the respective cluster cells plus the users obtained by considering the intersection with the missing values intervals. For an efficient generation of cluster approximations, we partition a set of users U_A according to the amount of missing values per user, i.e., $(U_A, S) = ([U_A^0, U_A^1, \cdots, U_A^x], S)$, where U_A^i is the users with exactly i missing values. To avoid analyzing all possible subspaces, the monotonicity of approximations and the fault tolerance thresholds are exploited. To generate the actual clusters from the approximations, the approximation list of users is traversed and users are added to the cluster. Users with no missing values (positioned in the beginning of the list) are added to the cluster, whereas users with missing values are gradually added to the cluster if they do not violate the fault tolerance thresholds ϵ_u, ϵ_i and ϵ_g.

To generalize, through subspace clustering, we receive like-minded users for a query user u, from several subspace clusters whose u is member of. Combining all users from the different subclusters u belongs to, is advantageous, as we gain an extensive selection of like-minded people for u, which might be based on different subsets of items. Thus, we are able to reflect on different characteristics u might feature in order to calculate the most promising recommendations for him/her.

4. EXPERIMENTAL EVALUATION

In this section, we evaluate our approach using a MovieLens dataset[1] that contains 100,000 ratings given by 983 users for 1,682 movie items (*ML-100K*). We compare the following approaches: (i) The *naive* approach that performs a sequential scan of the database to retrieve the set of friends for a user. (ii) The *fullClu* approach that organizes users into clusters using a hierarchical clustering algorithm in the full item space I. The friends of a user are his/her cluster members. (iii) The *gridFTSC* approach that organizes users into possibly more than one clusters defined upon different subspaces of items.

We evaluate both the efficiency and quality of recommendations achieved by the different algorithms. Regarding *efficiency*, we study how the runtime of the algorithms is affected by the different parameters. To evaluate the *quality* of user recommendations, we use predictive accuracy measures, namely, MAE and $RMSE$, that directly compare the predicted user ratings with the actual user ratings. MAE (Mean Absolute Error) signifies the average of absolute errors of the predictions compared to the actual given ratings for a user, whereas $RMSE$ (Root Mean Squared Error) expresses the average of squares of absolute errors. The smaller these values, the better the quality of the recommendations. We also study the *number* and *size* of generated clusters as an indirect measure of quality for the different approaches. The intuition is that when a user belongs to a very small cluster, his/her friends selection is limited and therefore the recommendations are worse comparing to a user that gets recommendations from a larger pool of friends.

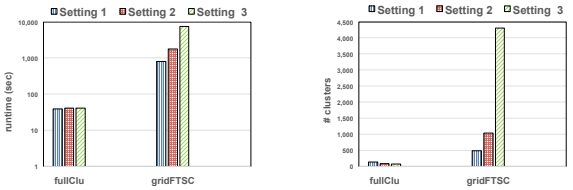
Experiments run on a 2.5 GHz Quad-Core i5-2450M architecture with 8.00 GB RAM and a 64-bit operating system. The distance between two users is evaluated as the Euclidean distance over their commonly rated items. When a specific subspace is considered, the distance relies only on the items that comprise the subspace.

4.1 Parameter Settings and Efficiency

We study the execution time of the different methods under different parameter settings, as well as the number and dimensionality of the resulting clusters. The execution time is a performance measure, whereas the number of resulting clusters provides some hints on the quality of the recommendations. We do not report on the naive approach here; according to our previous work [4], it takes four times longer than the full clustering approach. Regarding fullClu, the smaller the number of clusters, the better, since a small number of clusters might be an indication of clusters of large cardinality and this is exactly what we need to receive a broad selection of friends. On the contrary, for gridFTSC, a good clustering for recommendations is one featuring many subspace clusters. This is because each user potentially belongs to several clusters, which again offers a wider selection of friends.

We experimented with different parameters, an overview is shown in Table 1. We assign unique identifiers (IDs) to the parameter settings for each approach and use both approach name and parameter settings ID hereafter to denote the approach and the selected parameters. The runtime and the number of generated clusters for each approach and setting is depicted in Figure 1.

[1] `http://www.grouplens.org/node/73`



(a) Runtime (b) # Clusters

Figure 1: Runtime (in logarithmic scale) and #clusters for different parameter settings (c.f. Table 1).

fullClu: δ has a strong impact on the algorithm performance. The higher the value of δ, the lower the number of generated clusters and therefore, the bigger (in average) the resulting clusters. This is expected since more clusters get merged by the hierarchical clustering when δ is relaxed. For $\delta = 0.2$, the cluster population is in the [3-15] users per cluster range and there are three big clusters containing 27, 38 and 96 users, respectively. For $\delta = 0.5$, the range is [3-27] and there is one big cluster of 138 users. For $\delta = 0.7$, the range is [3-28] and the big cluster of 138 users still exists. The execution time also depends on δ, the more clusters are merged the longer the algorithm runs.

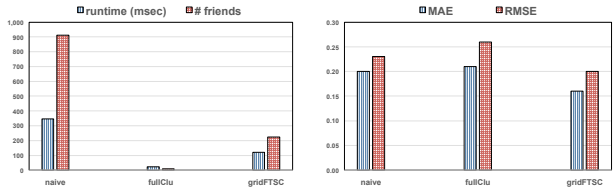
Comparing to the other approaches, the number of generated clusters in fullClu is the smallest. While the runtime of the algorithm is the best compared to the other approaches, the clustering result might be problematic for determining users' friends. For example, if we want to find the friends of a user belonging to one of the (many) small clusters, the friends selection is very limited and therefore, the quality of the recommendations might be poor. The problem might be not so severe for a user belonging to a (usually one) big cluster, as his/her friends selection is more broad.

gridFTSC: The lower the density threshold $minPts$, the larger the number of generated clusters and the more higher-dimensional the derived clusters are. This is due to the fact that with a lower density threshold more grid cells are considered as dense. For the same reason, the runtime increases with a decreasing value of $minPts$, as the number of clusters and the number of items to be considered for extension increases. For $minPts = 50$, the algorithm detects 494 subspace clusters, whose dimensionality lies in [1-4] range. The higher-dimensional subspace clusters are based mostly on the same subset of items and just differ in one or two additional ones. This implies that the dataset features some prominent items, which "derive" big clusters, like items 1, 12, 50, 98, 100, 127, 172 and 174. Many of the detected clusters, include these items. A closer inspection of the dataset confirms our observation: movies corresponding to these ids received 300 - 400 ratings compared to considerably less ratings given for other movies (about 50 - 100). Also, many 1-item subspace clusters were detected, which could not be extended to a higher dimensionality due to violation of the density threshold *minPts*. When we lower the threshold, the number of clusters as well as the running time increase drastically, however the maximum subspace dimensionality of the clusters not; the maximum dimensionality is 5 for both $minPts = 40$ and $minPts = 50$. Higher-dimensional subspace clusters are desirable, as they demonstrate a higher agreement in terms of preference of the included users. Therefore, the choice of a value for *minPts* is a trade-off between algorithm run-time and clustering result quality. Comparing to the other approaches, gridFTSC

Table 1: ML-100K dataset: Parameter settings and results

	Parameter settings	Runtime	# Clusters	Setting ID
fullClu	$\delta = 0.2$	39s 585ms	141	1
	$\delta = 0.5$	41s 115ms	87	2
	$\delta = 0.7$	41s 237ms	83	3
gridFTSC*	minPts = 50, grid = 3	13min 33s 55ms	494	1
	minPts = 40, grid = 3	29min 53s 364ms	1038	2
	minPts = 30, grid = 3	2h 3min 41s 655ms	4308	3

(*) parameters for gridFTSC: $\epsilon_o = 0.4$, $\epsilon_s = 0.3$, $\epsilon_g = 0.4$



(a) Runtime & #friends (b) MAE & RMSE

Figure 2: Top-10 user recommendations statistics (ML-100K dataset, Setting 1 from Table 1).

generates the larger number of clusters and therefore, it is the slowest method.

To conclude, although fullClu is fast, the clustering quality is questionable as it consists of several very small clusters, which might lead to poor recommendations for the query users. On the contrary, gridFTSC results in many subspace clusters and a large subspace variety.

4.2 Quality of User Recommendations

Here, we examine the qualitative differences of the different approaches. We select specific users having different demographics (occupation, age range and sex) and a number of ratings equal to the average number of ratings per user, and compute recommendations for all different approaches. We chose to issue the 10 most promising (top-10) recommendations to the respective user. For the naive approach, the distance threshold of the fullClu approach was employed. Due to lack of space, we report here on a single user.

The selected user is a 34-year old educator, who has submitted 70 ratings. According to his/her ratings, he seems to be interested in genres like Drama (30 ratings), Action (18 ratings), Comedy (18 ratings) and Romance (14 ratings), whereas he usually does not watch Documentaries, Fantasy, Horror or Western movies (0 ratings). For recommendation calculations, we employed the parameters settings 1 (c.f., Table 1), which give the less promising clustering results for all different approaches. For fullClu, setting 1 results in the smallest clusters and therefore, in a limited choice of friends. For gridFTSC, this setting also results in the lower number of generated subspace clusters and the lower number of considered dimensions for these clusters (Figure 2).

The runtime is a great deal lower when employing user clusters. The naive approach scans the whole database to determine the friends of the query user. Moreover the result set is big (912 out of 942) due to the high distance threshold. fullClu is the fastest method by far, however its quality of predictions suffers heavily from the small set of friends considered. The set of friends generated by gridFTSC is larger than the one generated by fullClu but still small compared to the one generated by naive. MAE and RMSE, however, show that the quality of the predictions by gridFTSC increases when compared to naive. This is due to a more careful selection of friends. gridFTSC is slower than fullClu but much faster than naive. With respect to the naive approach, gridFTSC agrees on 7 out of 10 recommended items, whereas fullClu shares only 3 items with the other approaches. This poor performance is due to the narrow selection of friends, as our user did belong to a small cluster.

To conclude, the fault tolerant subspace clustering approach seem to overcome the friends selection problems of naive (very broad) and fullClu (very narrow) by providing a wider selection of diverse friends at a reasonable runtime.

5. CONCLUSIONS

In this paper, we integrate subspace clustering into the recommendation process. Our experiments show that the fault tolerant subspace clustering approach outperforms both the *naive* approach that scans the whole database to derive similar users and the $fullClu$ approach that pre-computes full dimensional clusters of users and relies upon these clusters for the recommendations. This suggests, that the diversity in the results due to subspace clustering has a positive effect on the recommendation process.

In our current work, we study how to explore density-based subspace clustering approaches in order to detect arbitrarily shaped clusters, that seem more appropriate for our problem. Interestingly, we can adapt the candidate approximations construction by using density-based clusters, instead of considering grid-cells. The difficulty that emerges when transferring fault tolerance to density-based subspace clustering is that density-based approaches use a distance function and so, we need to evaluate distances between users, even though they might contain missing values.

6. REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

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

[3] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, Mar. 2009.

[4] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In *ER*, 2012.