Building Recommendation Systems using Peer-to-Peer Shared Content

Yuval Shavitt
School of Electrical Engineering
Tel-Aviv University, Israel
shavitt@eng.tau.ac.il

Ela Weinsberg
Dept. of Industrial Engineering
Tel-Aviv University, Israel
ela@eng.tau.ac.il

Udi Weinsberg
School of Electrical Engineering
Tel-Aviv University, Israel
udiw@eng.tau.ac.il

ABSTRACT

Peer-to-Peer (p2p) networks are used for sharing content by millions of users. Often, meta-data used for searching is missing or wrong, making it difficult for users to find content. Moreover, searching for new content is almost impossible. Recommender systems are unable to handle p2p data due to inherent difficulties, such as implicit ranking, noise and the extreme dimensions and sparseness of the network.

This paper introduces methods for using p2p data in recommender systems. We present a method for creating content-similarity graph while overcoming inherent noise. Using this graph, a clustering method is presented for detecting proximity between files using the "wisdom-of-the-crowds". Evaluation using songs shared by over 1.2 million users in the Gnutella network, shows that these techniques can leverage p2p data for building efficient recommender systems.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms, Similarity measures*

General Terms

Algorithms

1. INTRODUCTION

Peer-to-Peer (p2p) networks are used for sharing content (i.e., files) by millions of users, making them an extremely valuable resource for information retrieval tasks. Searching for content is mostly done using query strings that are matched against meta-data fields attached to the content, e.g., song files contain tags that describe the name of a song, artist, genre, etc. Often, some of this data is missing, incorrectly spelled or encoded (such as musical genre) making it difficult for users to find the data they are looking for in the abundance of existing content. Indeed, it was found thatonly 7–10% of the queries are successful in returning useful content [7] in Gnutella [4].

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

Various techniques were suggested for improving the accuracy of information retrieval in large-scale p2p networks. However, current p2p networks still employ simple string matching algorithms against file name and meta-data. Recommender systems were also suggested to help users find new content based on their preferences or similarity to other like-minded users. These systems have been studied extensively in recent years [3], mostly relying on the willingness of users to rank their preferences in order to provide better recommendation. However, p2p networks, alongside with new home entertainment services such as IPTV, introduce new difficulties to standard recommender systems making them somewhat not fitted for this task.

First, users do not explicitly rank files, but simply download them. This *implicit ranking* makes it difficult to assess whether a user "likes" the downloaded content. Second, there is a large amount of noise that inherently exists in p2p networks, since content is inserted to the network and labeled (tagged) by the users. This results in an abundance of duplicate content, multiple and even conflicting tagging and spelling mistakes or ambiguities. Finally, p2p networks are used by many users sharing lots of content, whereas a given user holds only a tiny fraction of the content, resulting in extreme sparseness of the user-to-content matrix.

These complexities often render most recommender systems useless for helping users find new unfamiliar content, as they cannot efficiently leverage data that can be extracted from p2p networks. The objective of this work is to overcome the above difficulties and present methods that enable recommender systems to leverage p2p data. This, in turn, can improve the ability of users to find content, contributing to an overall improved user experience.

The contribution of this work is twofold: (a) we present a method for incorporating large-scale p2p data into recommender systems, by clustering the shared file similarity graph, and (b) an in-depth study of a snapshot of the content shared by over 1.2 million users in Gnutella is performed, used for evaluation of the proposed clustering technique.

2. METHODOLOGY

2.1 P2P Network Model

The p2p network is modeled as a bipartite graph that connects users to their shared content (files). Notice that this graph is a special case of the standard collaborative filtering matrix in which a link in the graph represents the ranking of an item by a user. This graph is transformed into a 1-mode file-similarity graph, S, where the weight of a link

between two files is the number of users that share them. Additionally, a file popularity vector P is created, counting, for each file, the total number of users that share it. In order to account for high variance in the popularity of files, the link weight, w_{ij}, is normalized using the cosine-distance $\widehat{w}_{ij} = w_{ij}/\sqrt{P_i \cdot P_j}$.

This graph alone is already quite useful for building recommender systems as it provides a distance between any two files, in a way that captures the "wisdom of the crowds". Although this method requires crawling the network, it is significantly less computationally expensive than signal processing methods that are commonly used for estimating distance between media files. Additionally, the similarity graph preserves user privacy, since although it is constructed using private data regarding shared files, this data is aggregated from a large set of users, therefore any data regarding a specific user is extremely hard to extract. A second usage of this graph is for extracting distance between users based on their shared content [5] which helps overcome the low overlap of content between any two arbitrary users.

2.2 Clustering Shared Content

Clustering is often used as a method for extracting similarity between files [6]. We design a clustering algorithm, which is similar to the Partitioning Around Medoids (PAM) or k-medoids [2] method (a variation of k-medoids). The algorithm, Graph k-Medoids (GkM), receives as input the file similarity graph, $\widehat{S}(V, E)$, the number of output clusters, k, and a balancing threshold τ. A function $d(w)$ is used to convert similarities to distances. The distance between two connected vertices is the sum of distances along the shortest path, e.g., using Dijkstra algorithm.

GkM first iteratively selects a set O of k vertices that are used as the cluster origins. The origins are selected such that the distance between any two origins O_i, O_j is at most min_dist. The value of min_dist is calculated by $min\_dist = \gamma \cdot d_{max}/k$, for a given $0 < \gamma < 1$. The maximal distance, d_{max}, is heuristically calculated by randomly selecting a vertex, finding the most distant vertex from it, and then finding the most distant vertex from the latter. The minimal distance ensures that clusters will not overlap, resulting in possible mixture of their features. γ relaxes the d_{max}, providing high probability of selecting cluster origins that are distant enough.

Once the cluster origins are selected, the algorithm scans all the vertices and assigns each vertex to the cluster whose origin is the nearest. In case several cluster origins have similar distance up to the given threshold τ, the vertex is assigned to the smaller cluster, hence heuristically balancing the cluster size. Since only the distance to each origin is used, only k single-source shortest paths are performed hence any further distance calculations are avoided.

GkM performs a single iteration on the vertices and does not change the origins when assigning vertices to clusters, hence avoiding excessive shortest-path calculations. However, this causes GkM to not guarantee convergence to a local minima, which is a key strength of k-means. The size balancing threshold, τ, serves as a weak optimization for uniformly spreading vertices among clusters, since it shifts only border-line vertices and performs it inside the single iteration. However, as we later show despite the single iteration and relaxed distance and size optimizations, the algorithm manages to uniformly partition the input graph.

Due to the random nature of the selection of cluster origin, it may be thought that the resulting clusters may vary significantly between runs. However, the heuristic behind the algorithm design is that due to the large size of the graph, the min_dist limitation and existence of popular and strongly connected vertices, clusters quickly "grow" towards popular vertices and extend from there.

3. EVALUATION

3.1 Dataset

The clustering algorithm is evaluated using a snapshot of the music files that are shared in the Gnutella [4] p2p network. The files were collected by a 24 hours active crawling of the shared folders of over 1.2 million users on the 25th November 2007, selecting only files that correspond to musical content (.mp3 files). Using this data, a song similarity graph was created, having songs as vertices and the weight of a link connecting two vertices is the total number of different users that shared both files. The graph has over 531k songs and more than 1 billion links. Songs are indexed by descending order of their popularity.

Weak links between songs, that are due to malicious and spam files or represent scarce relations, are filtered out to reduce bias. Keeping for each file only the top 40% links (ordered by descending similarity value) and not less than 10 keeps over 20 million undirected links.

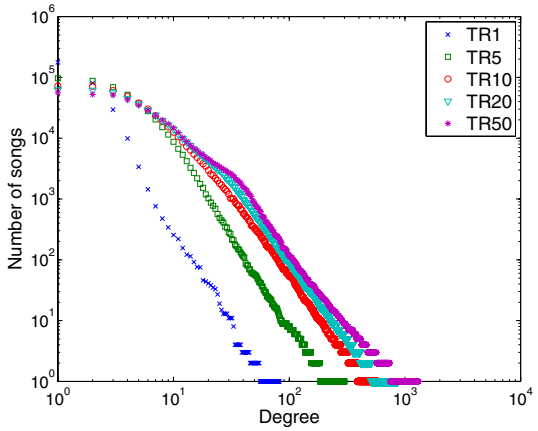
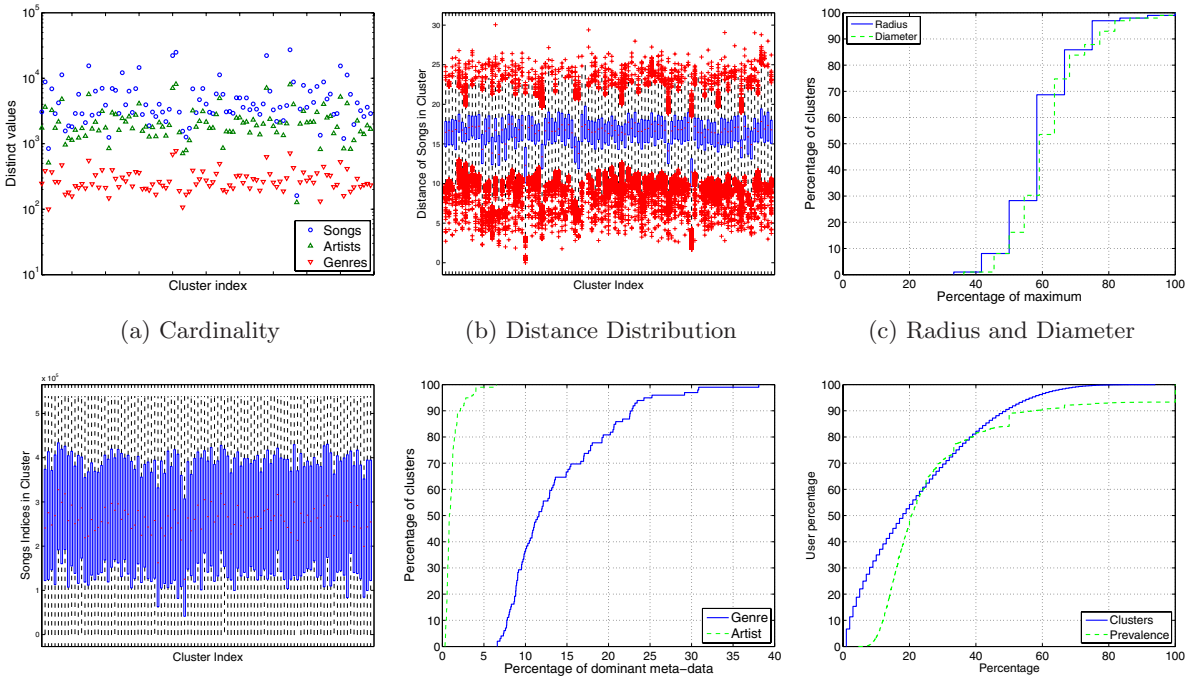


Figure 1: Degree distribution using sampled graphs

The graph is still extremely large and sparse, having less than 0.03% of possible links (NetFlix, which is considered very sparse, has 1% of the possible links exist). Therefore, we create smaller sub-networks that contain, for each vertex, only the top N neighbors, ordered by decreasing normalized similarity. In order to make sure that these sub-networks do not significantly change the graph, we plot the degree distribution for values of N. Fig. 1 shows that while for N=1 the distribution is extremely sparse, for $N \geq 10$ the degree distributions are almost identical with slightly higher node degrees as N grows. Notice that the power-law distribution suggests that there are relatively a few songs with very high connectivity and many songs with low connectivity.

Most music files contain meta-data that provides information about the song, such as its name, performing artist, genre etc. Before analysis, all coded genres were resolved



(a) Cardinality (b) Distance Distribution (c) Radius and Diameter

(d) Popularity (e) Metadata Prevalence (f) User Mapping

Figure 2: Efficiency measures of the clusters resulting from GkM, using k=100 and TR10

into the appropriate genre strings. Overcoming spelling mistakes that exist in over 20% of the meta-data in the Gnutella network [7] is achieved by using SoundEx.

Analysis of the genre reveals that over 35% of the files are missing a genre, and the remaining files have over 3600 different genres, with the top being rap (9%), rock (8.8%) and pop (4.5%). Over 14% of the files are missing an artist tag but the remaining span across over 100,000 different artists.

3.2 Clustering Efficiency Measures

Since the goal of the clustering process is to improve ability to recommend content from p2p networks, we present efficiency measures that aim to answer two questions: (a) how closely related is the content in each cluster, and (b) how different is the content between any two clusters. Since answering these questions is highly subjective [1], we define a set of quantifiable efficiency measures. Evaluation is performed using the graph $TR10$ with k=100. Converting similarity to distance was performed using $d(w) = -\log_2(w)$.

3.2.1 Cardinality and Meta-data

A uniform distribution of songs into clusters is desired but unlikely since musical content has a few main-stream genres and many niches. Main-stream clusters are expected to be much larger than the ones that represent a niche. Fig. 2(a) plots the number of songs, distinct genres and distinct artists in each cluster. The figure shows a relatively balanced distribution of songs, genres, and artists among the clusters. The median cardinality is roughly 3k songs, but 4 clusters have over 20k songs, each. The lower the number of genres and songs relative to the number of songs, the more accurate is the identification of musical preferences, hence better clustering results.

3.2.2 Distance Distribution

Unlike k-means, GkM cannot guarantee convergence to local minima. Fig. 2(b) depicts the distribution of the distances of all files in each cluster from the corresponding cluster origin on a boxplot (the edges mark the 25th and 75th percentiles and points beyond that are marked as outliers). The figure shows that GkM creates clusters that mostly have a uniform distribution with overlapping distances of the major percentiles. Even large clusters that contain more distant outliers have a similar distance distribution. The heavy distribution of low-distance outliers is due to the way the cluster "grows" while collecting near-by vertices.

3.2.3 Radius and Diameter

The radius of a graph is the minimum over the maximum shortest paths between a node and all other vertices and the diameter is the maximum distance between any pair. We normalize these with the maximal value across all clusters. The distribution can help assess whether most clusters are relatively "tight" or are spread in a large space. Fig. 2(c) shows the cumulative distributions of the radius and diameter percentages. GkM results in relatively uniform cluster spreading, having 80% of the clusters be 50-70% of the maximum cluster, which aligns with the distance distribution.

3.2.4 Diversity of Popularity

For the clusters to be efficiently used by recommender systems, it is important that each cluster contain files with large diversity of popularity. This allows a recommender system to recommend less popular content, which is often difficult to find. Fig. 2(d) shows that almost all clusters hold a mixture of popular and unpopular songs. Furthermore, most clusters have at least one song from the top 2,500.

3.2.5 Meta-data Prevalence

Having files in a cluster sharing various features is expected to be reflected in their meta-data. For each cluster, we find the dominant genre and dominant artist (i.e, those that have highest prevalence), and their prevalence.

Fig. 2(e) plots the CDF of the percentage of dominant genres and artists. The figure shows that in roughly 60% of the clusters more than 10% of the songs belong to the dominant genre, and in 5% of the clusters 30% of the songs belong to the dominant genre. This shows that many of the songs in a cluster share a common feature (recall that there are over 3600 genres in our database). Additionally, in 94% of the clusters the maximal prevalence of the dominant artist is less than 3%. However, noting that there are over 100k artists in the dataset, it is still quite high.

3.2.6 Meta-data Overlap

We counted the number of clusters in which each of the 10 most dominant genres (Rock, Rap, Latin, Pop, Blues, Metal, Chanson, Country, R&B and j-Pop) appear as dominant. As can be expected by looking at the very generic genre names, we found that a few genres are dominant in many different clusters, having "Rock" being dominant in 37% of the clusters. This reinforces the claim that there can be many "flavors" of the same genre which are not properly tagged but are successfully identified using cluster analysis. The distribution of dominant artists is, as expected, much broader than genres, where only a few artists appear as dominant in more than one cluster. For example, although Lil Wayne's songs are tagged with 4 genres (Rap and Hip-hop in our database and Rock and Pop-Rap in Wikipedia), he appears in more than 10 different clusters.

4. VALIDATION

As a real world application, the song clusters are used for creating a recommendation system that provides users with songs that are related to the ones they already share. Since people are assumed to have a defined taste in music, it is expected that the songs of each user will reside in a small number of clusters with a large fraction of the songs in one of the clusters (the *dominant* cluster). Using the original user-to-song network, we count, for each user, the number of songs in each cluster. Fig. 2(f) plots the CDF of the percentage of clusters that each user has songs in and the CDF of the percentage of songs (prevalence) each user has in the corresponding dominant cluster. The figure shows that 11% of the users have all their songs in a single cluster and almost 70% are mapped to less than 10 different clusters. The median value of songs in the dominant cluster is almost 70%. The steep jumps in the 100th percentile and in 50%, 67% and 33% are mostly due to users that have only a few songs (less than 10).

Once clusters are obtained, creating recommendations to a user is achieved by suggesting the songs that are nearest to her known songs within her dominant cluster. On each iteration, the algorithm recommends the nearest song, and repeats this process until no more recommendations are needed. Unlike collaborative-based recommendations systems that find like-minded users, this technique maps a user to a cluster using information spread in *all* users in the network. This global information leads to more accurate results than standard approaches.

Evaluating the correctness of the results is done using a training set of 30% of the songs of each user. Then, the recommender system attempts to recommend exactly the remaining 70% of the songs the user has. We evaluate by resolving the artists and see how many of the artists in the evaluation set appear in the recommendation.

Evaluation using TR20 and k=100 results in a median precision (correct recommendations out of the recommended set) of 12.1% and recall (correct recommendations out of the real data) of 12.7%. These results are quite good considering the vast amounts of songs that exist and shared by p2p users. For example, one user in our dataset had songs tagged with "Bob Dylan ft. Van Morrison", "Chuck Berry" and "Bob Dylan". Given only one song which was tagged with "Bob Dylan ft. Van Morrison", our algorithm recommended songs that are tagged with "Van Morrison". Although this is clearly a good recommendation, this recommendation was not counted as a "hit", since none of the recommended songs matched the evaluation set. This shows that it is possible, yet not trivial, to recommend songs or even objectively assess the recommendations.

5. CONCLUSION

This paper aims to bridge the gap which exists in using large-scale p2p data for building recommendation systems. We propose two methods for improving recommendation systems – estimating file distance without heavy calculation, and content clustering. The observations made on the coherency of the clusters enable us to develop a straightforward recommender system that captures the complete network "knowledge" by utilizing the clusters. Evaluation performed on data from Gnutella, using data from over 1.2 million users, shows that the proposed methods can leverage p2p data for building efficient recommender systems.

Acknowledgment. This research was supported in part by the Israel Science Foundation (ISF) center of excellence program (#1685/07) and by the Ministry of Trade and Industry, MAGNET program through the NeGeV Consortium.

6. REFERENCES

[1] G. Geleijnse, M. Schedl, and P. Knees. The Quest for Ground Truth in Musical Artist Tagging in the Social Web Era. In *ISMIR*, Vienna, Austria, Sept. 2007.

[2] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. Wiley Interscience, 1990.

[3] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3), 1997.

[4] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *First International Conference on Peer-to-Peer Computing*, 2001.

[5] Y. Shavitt, E. Weinsberg, and U. Weinsberg. Estimating peer similarity using distance of shared files. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2010.

[6] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD*, 2000.

[7] M. A. Zaharia, A. Chandel, S. Saroiu, and S. Keshav. Finding content in file-sharing networks when you can't even spell. In *IPTPS*, 2007.