Community Structures in Recommender Systems

Ehsan KAZEMI
LCA4, I&C, EPFL

***Abstract*—In the age of information overload, recommender systems help users to find what they like, but in return they can affect users interests. Recommender systems can narrow users options to a limited community of items or informations. Our goal is to develop tools to investigate the effect of recommender systems on the social networks. Netflix Prize winner algorithm is an example of good recommender system to be studied. Netflix Prize winner is built based on the combination of three different contributions. We explain one of the major contributions involved in the winner algorithm. Community detection algorithms and clustering functions provide a powerful tool to analyse effect of recommender systems on the networks. Community detection algorithms literature are briefly surveyed in this report. We should note that clustering functions have limitations in the detection of clusters. Finally, an inherent contradiction between properties that we expect intuitively from clustering functions is stated and proved.**

***Index Terms*—Recommender systems, Netflix prize, community detection, clustering function.**

I. Introduction

MARK Zuckerberg, Facebooks chief executive once told that "a squirrel dying in your front yard may be more relevant to your interests right now than people dying in Africa." One great business strategy is focusing on the most personally relevant informations. As a result of information explosion and fast growth of e-commerce technologies users are overloaded by huge amount of options and data to consider. A user may not have enough time or knowledge to process huge amount of information or evaluate all the available options. *Personalization* helps users to obtain the information which they are seeking for. *Recommendation systems* suggest items based on the costumer's preferences. It is not so exaggerated to say that "The Age of Search has come to an end" and we are entering the "Age of Recommendation."

The goal of recommendation systems is to capture user's interests and recommend other items that are more probable to be liked by that user. When a recommender system builds a user profile, it collects a set of implicit and explicit information about that user. Explicit data for example is gathered by asking a user to rate a set of items. By observing the set of items a user looked at or purchased, implicit data about user's interests is obtained. Also, communities that a user belongs to are a good source of implicit information about likes and dislikes of that user. A recommender system tries to generate meaningful recommendations to users based on the explicit and implicit data from their preferences .

There are three types of recommender systems: knowledge based recommenders, content filtering algorithms and Collaborative Filtering (CF) algorithms. If we want to predict how a user u rates for a new item i, CF method uses the ratings of users who share same interests with the user u or use the ratings of user u for items similar to the item i. CF is the most well known type of recommender system. Recommender systems capture users interests and recommend items based on their preferences. These recommendations can have influence on the users interests. We should pick a good recommender system to investigate its characteristics and study its effects on the evolutions of users/items communities. Netflix Prize winner algortihm is one good option for this propose. Netflix is a US online movie rental service that rents movies by mail. Its database contains 55 million DVDs of more than 100,000 movie titles. It recommends movies based on the recommender system called "Cinematch". Up to 60 percent of Netflix's rentals is a result of the personalized guesses that it can make about each user's movie interests. Netflix Prize was an open competition to design the best Collaborative Filtering (CF) algorithm to improve the accuracy of predictions about how much someone is going to enjoy a movie based on their previous preferences of movies. In section II we describe the winner algorithm of the Netflix Prize competition.

New items and users are added to the recommender systems. Communities of users and clusters of items evolve over time. Recommendations affect users interests over time. Predictions

Proposal submitted to committee: September 13th, 2011; Candidacy exam date: September 20th, 2011; Candidacy exam committee: Prof. Patrick Thiran, Prof. Matthias Grossglauser, Prof. Christina Fragouli.

This research plan has been approved:

Date: _____

Doctoral candidate: _____
(name and signature)

Thesis director: _____
(name and signature)

Thesis co-director: _____
(if applicable) (name and signature)

Doct. prog. director: _____
(R. Urbanke) (signature)

become more accurate if recommender systems apply users feedback about recommendations efficiently. One good tool to capture these changes, study their interactions and investigate their effects on the social network is to monitor communities of both users and items. Observing interactions between communities and clusters and analysing their evolution is a significant source of information about mechanism of recommender systems. We give a brief survey of community detection algorithms in section III .

Clustering is one of the most widely used techniques for exploratory data analysis, with applications ranging from statistics, computer science, biology to social sciences and psychology. Given a set of data points $x_1, x_2 \cdots x_n$ and some notion of similarity $s_{ij} \geq 0$ between all pairs of data points x_i and x_j, the intuitive goal of clustering is to divide the data points into several groups such that points in the same group are similar and points in different groups are dissimilar to each other. Clustering functions are one of available powerful tools for analysing the effect of recommender systems on the networks and communities of users or items. Also, it is possible to use clustering functions to improve efficiency of recommender systems. An inherent contradiction between the intuitive properties we expect from a clustering function is stated and proved in section IV.

II. The BellKor Solution to the Netflix Grand Prize [1]

The Netflix Grand Prize competition was held by Netflix. It was supposed that winner is an algorithm that improves the root mean square error (RMSE) of predictions more than other candidates with at least 10% of improvement over Netflix's recommender system, "Cinematch". It is useful to mention that there are evidences that small improvements on the RMSE values have significant effects on the quality of top recommended movies.

The Netflix data set contains more than 100 million movie ratings provided by $m = 480,189$ anonymous users for $n = 17,770$ movies. The rating values are integer numbers between 1 to 5. Data was collected in a period of 7 years. This data set is divided to three different categories. The first category is called Training set that is 99,072,112 ratings of movies in the form of $< user, movie, date\ of\ grade, grade >$. Second one is the Probe set of 1,408,395 ratings. The last category is a collection of 2,817,131 ratings of movies that their values are just known to judges. This set is randomly divided into two parts of equal sizes. The first one that is called Test set is used to evaluate the final submissions of the competition and another one, Quiz set, is used for calculating the leader board. The goal of competition was to design an algorithm that predicts unknown ratings of movies in Test set with lowest RMSE.

On September 21, 2009 Netflix awarded the prize to the "BellKor's Pragmatic Chaos" team. Winner algorithm is a combination of contributions from three different teams (Bellkor, Pragmatic Theory and BigChaos) with a Test RMSE of 0.8567. "BellKor's Pragmatic Chaos" team described their algorithm in three different papers. In this section we will explain the final contribution of "Yehuda Koren" to the Netflix Prize winner algorithm.

Having borrowed the notations from [1], users and movies are shown by letters like u, v and i, j respectively. The rating of user u for movie i is represented by r_{ui}. The scalar value t_{ui} indicates the time of rating r_{ui}. Each user typically rates for small portions of the movies so the vast majority of ratings in the Netflix dataset (99%) are unknown. The set of pairs (u, i) which values of r_{ui} are known for them in the training set is represented by $\mathscr{K} = \{(u, i)|r_{ui} \text{ is known}\}$. Set of all the items that ratings of user u is available for them is shown by $\mathrm{R}(u)$ and also set of available rating for movie i is shown by $\mathrm{R}(i)$. $\mathrm{N}(u)$ indicates all the movies that user u has provided ratings for them (those ratings that are available or not). Predicted value of r_{ui} is denoted by $\hat{r}_{ui}$. At the following we describe series of predictors that are involved in the final solution.

A. Baseline predictors

Sometimes there is a systematic tendencies for some users to give higher ratings than others and some movies are more eligible to obtain higher ratings than the other movies. This effects which are independent of user-item interactions could be modeled by *baseline predictors*. If overall average of ratings be equal to μ, value of baseline predictor, b_{ui}, is calculated by

$$b_{ui} = \mu + b_u + b_i \tag{1}$$

where b_u and b_i are deviations of user u and movie i from average respectively.

Baseline predictors should be time dependent to capture temporal effects in the model. Popularity of a movie may change over time. One good example could be change of a movie bias as a result of changing the popularity of that movie's actress by some other events like play in another movie. It is possible that users change their baseline ratings over time, i.e. users tend to give higher or lower ratings than before. Also it is possible that taste of users change over time. The modified version of baseline predictor is

$$b_{ui}(t) = \mu + b_u(t_{ui}) + b_i(t_{ui}). \tag{2}$$

There are two major differences between movies and users biases. The first difference is that for movie biases we do not expect to see fluctuations in the ratings on a daily basis, but daily changes is expectable for users. The second one is that there are more ratings for movies with respect to users. Based on these two differences we can easily model $b_i(t_{ui})$. We divide item biases into several time periods. That is enough to assign a fixed value to item biases in each of these time periods (bins).

$$b_i(t) = b_i + b_{i,Bin(t)}. \tag{3}$$

Since we have a few number of ratings for users and user's ratings are more time transient, it is not accurate enough to use the idea of time bins for modeling temporal effects in the users biases. One solution to this problem is to use a linear function to capture time dependency of user biases. Deviation

of user u at time of rating, t, from biases is modeled by

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^{\beta}, \quad (4)$$

where t_u is the mean date of ratings by user u. For each user u we define a parameter α_u that is multiplied by the deviation in eq. (4) to model *gradual drifts* of user biases. These deviations model gradual drifts of user's tendencies. Sometimes there are *sudden drifts* in the users tendencies, such as bulk ratings at a specific day. It is possible to model these sudden drifts of user u at day t by b_{ut}. Concluding all the above discussions, for user biases we reach to the following equation.

$$b_u(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{ut}. \quad (5)$$

It is possible that each user responses to movie biases with different scales, so we multiply movie biases $b_i(t)$ by a user-time dependent scale of $c_u(t)$.

Interestingly, number of ratings a user gives in a specific day will influence movie biases. The reason for this fact is that when a user rates in a massive way it is more probable for her to rate movies she liked or hated more than other movies, because it is easier for a user to recall those movies. We assume that when users rate in a bulk, they still reflect their normal preferences. Accordingly we add another term $b_{i,f_{ui}}$ to baseline predictors for capturing frequency effects. The general form of baseline predictors with considering temporal effects would be

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,Bin(t_{ui})}).c_u(t_{ui}) + b_{i,f_{ui}}. \quad (6)$$

B. User-Movie Interactions [1], [4]

Using a very accurate baseline predictors is not enough to have a good personalized recommendations, because it dose not use any interaction between users and movies. One way to model user-movie interactions is to use *matrix factorization with temporal dynamics*. In matrix factorization model, users and movies are mapped to a f dimensional joint latent factor space. Ratings are modeled by an inner product in this f dimensional space. We assign to each user u and movie i two vectors $p_u, q_i \in \mathbb{R}^f$ respectively. The rating of user u for movie i is calculated by

$$\hat{r}_{ui} = q_i^T p_u. \quad (7)$$

In additions to rating values of movies, the dataset contains a list of the movies rated by users, regardless of how they rated these movies. It is possible to incorporate these implicit data in to matrix factorization model. Now each user u is modeled by $p_u(t) + \left|\text{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{N}(u)} y_j$. Parameter $p_u(t)$ models explicit ratings of user similar to the role of p_u in eq. (7). The model is complemented by $\left|\text{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{N}(u)} y_j$ which models implicit feedback of ratings. It was shown that incorporating implicit data in this model improves accuracy significantly.

Equation 7 or its modified version gives a good model for capturing user-movie interactions, but much of ratings are independent of these interactions and are just due to the user (movie) effects independent of movies (users). These effects were modeled by baseline predictors before. SVD++ is combination of baseline predictors and modified matrix factorization model.

$$\hat{r}_{ui} = b_{ui} + q_i^T \Big(p_u + \left|\text{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{N}(u)} y_j \Big). \quad (8)$$

In eq. (8) b_{ui} can be replaced by time changing baseline predictors of eq. (6).

It is possible to improve matrix factorization model by the ideas from the time changing baseline predictors section. Each one of the f elements of time dependent user factor, $p_u(t)^T = (p_{u1}(t), \cdots, p_{uf}(t))$, is modeled like the user biases in the eq. (5).

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \text{dev}_u(t) + p_{ukt}. \quad (9)$$

In this equation, p_{uk} represents the stationary bias of the factor, $\alpha_{uk} \cdot \text{dev}_u(t)$ approximates gradual drifts of factors and p_{ukt} absorbs short lived effects.

As we discussed earlier frequency biases should be associated with the movies, so we incorporate frequency awareness in to the movie factors by parameter $q^T_{i,f_{ui}}$. The new generalized version of movie predictors is

$$\hat{r}_{ui} = b_{ui} + (q_i^T + q^T_{i,f_{ui}}) \Big(p_u(t_{ui}) + \left|\text{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{N}(u)} y_j \Big). \quad (10)$$

Neighbourhood based CF algorithms absorbed widespread popularity because their understandability of reasoning behind computed recommendations and their applicability for new entered users and ratings. The static neighbourhood model without considering temporal effects is based on the following prediction formula:

$$\hat{r}_{ui} = b_{ui} + \left|\text{R}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{R}(u)} (r_{uj} - \tilde{b}_{uj}) w_{ij} + \left|\text{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \text{N}(u)} c_{ij}. \quad (11)$$

In eq. (11) w_{ij} and c_{ij} represent required adjustments for prediction of rating of movie i given the rating of movie j. Parameters w_{ij} model explicit data, i.e. interaction between movie-movie ratings. Implicit data, those disregard rating values and just consider only items were rated, is modeled by parameters c_{ij}. To address temporal dynamics we use eq. (5) for b_{ui}.

Values of w_{ij} and c_{ij} depend on the inherent movie-movie characteristics and they do not change over time. On the other hand, strength of parameters, w_{ij} and c_{ij}, depends on the difference between ratings time of movies i and j. For example, rating of movies i and j by user u in a short time period pushes w_{ij} and c_{ij} to higher values. On the other hand, if time periods between ratings be far apart they have low impact on each other. One reason for this model could be change of user taste during time. Based on the above discussions, movie-movie interaction weights effect could be modeled by an exponential decay function $e^{-\beta_u \cdot \Delta}$ with positive values of

β_u. Hence values of β_u are user dependent, i.e. some users are more consistent than other users and their taste change with lower rates (smaller values of β_u). The combination of new neighborhood based CF algorithm with baseline predictors that model temporal effects is

$$\hat{r}_{ui} = b_{ui} + \left|\mathrm{N}(u)\right|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} e^{-\beta_u \cdot \left|t_{ui} - t_{uj}\right|} c_{ij} + \left|\mathrm{R}(u)\right|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot \left|t_{ui} - t_{uj}\right|} ((r_{uj} - \tilde{b}_{uj}) w_{ij}). \quad (12)$$

Testing the described predictors on the Netflix dataset gives us the evidence that modeling temporal dynamics can have more effects on the efficiency of recommender systems than designing more complex learning algorithms, because users and movies attributes change continuously over time. In the following we describe predictors based on the restricted Boltzmann machine.

C. Restricted Boltzmann Machine [5]

Assume that we have M movies and N users. We define a Restricted Boltzmann Machine (RBM) with visible unit $\mathbf{V}$ and F hidden binary units ($\mathbf{h}$) for each user u. Every RBM has the same number of hidden units, but an RBM only has visible softmax units for the movies rated by the user. Here m represents the number of movies rated by the user u. Let $\mathbf{V}$ be a $5 \times m$ observed binary indicator matrix with $v_i^k = 1$ if the user rated movie i as k and 0 otherwise. Each column of the observed visible binary rating matrix $\mathbf{V}$ is modeled by a conditional multinomial distribution. Also, we can use a conditional Bernoulli distribution for modeling hidden units, $\mathbf{h}$. So we have

$$p(v_i^k = 1|\mathbf{h}) = \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)} \quad (13)$$

$$p(h_j = 1|\mathbf{V}) = \sigma(b_j + \sum_{i=1}^{m} \sum_{k=1}^{K} v_i^k W_{ij}^k) \quad (14)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic function. Each visible units of RBM corresponds to a specific movie that is rated by user, therefore b_i^k its the bias of rating k for movie i. Also, the bias of hidden feature j is represented by b_j.

Intuitively bias of RBM visible units models user biases, but in previous sections we explained that there are other important types of biases, user-bias (b_u^k), single-day user bias (b_{ut}^k) and frequency-based movie bias (b_{if}^k). Model would be more complete if we incorporate these biases by modifying conditional multinomial distribution of eq. (13). We can add a user/date bias term to the distribution equation

$$p(v_i^k = 1|\mathbf{h}, u, t) = \frac{\exp\left(b_{ut}^k + b_u^k + b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(b_{ut}^l + b_u^l + b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)} \quad (15)$$

In addition, for considering the significant effect of frequencies it is possible to condition on them:

$$p(v_i^k = 1|\mathbf{h}, u, t, f) = \frac{\exp\left(b_{if}^k + b_{ut}^k + b_u^k + b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(b_{if}^l + b_{ut}^l + b_u^l + b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)} \quad (16)$$

So far we have explained the Yehuda Koren's contribution to the Netflix Prize winner algorithm. As we mentioned before, the final solution to the Netflix competition was a coalition of three different teams. Although, try to uncover new informations from the data set by designing new models is a good way to improve the performance of predictors, blending predictors of different teams is a key idea to achieve highly competitive results on the Netflix dataset. It is possible to use Gradient Boosted Decision Trees (GBDT) to blend different predictors. This blending technique is applied to three distinct sets of 454 predictors of BellKor's Pragmatic Chaos team, 75 predictors which BigChaos selected them from the first 454 set of predictors and another set of 24 predictors that are built based on the above explained models in this report.

III. Community Detection in Graphs [2]

In this section, we study structure of communities in graphs. There are a lot of different definitions for a community in the literature and no definition is universally accepted by researchers. Definitions vary based on the application of community detection problem. Intuitively we expect that number of edges inside the community must be more than number of edges connecting community's vertices to vertices from other communities.

Assume that graph $\mathcal{C}$ with $n_{\mathcal{C}}$ vertices is a subgraph of graph $\mathcal{G}$ with n vertices. k_v represents degree of vertex $v \in \mathcal{C}$. Intra-cluster density ($\delta_{int}(\mathcal{C})$) equals to ratio of number of internal edges of subgraph $\mathcal{C}$ and number of all the possible edges between vertices in the subgraph $\mathcal{C}$. Similarly, inter-cluster density ($\delta_{ext}(\mathcal{C})$) is defined as the ratio of number of external edges of subgraph $\mathcal{C}$ and number of all the possible edges between vertices of $\mathcal{C}$ and all the other outside vertices. Large values of intra-cluster density with respect to the average link density of graph $\mathcal{G}$ ($\delta(\mathcal{G})$) and smaller values of inter-cluster density make subgraph $\mathcal{C}$ more like a community.

There are two ways to categorize community definitions in the graphs: local and global definitions. Local definitions consider a subgraph and its neighbours independent of the whole graph and try to specify communities based on their local properties. One possible definition is based on the reachability of vertices. If number of paths between two vertices is large, with higher probability we can say that they are in the same community. It is possible to define a community as a set of vertices that their pairwise distances, i.e. shortest path between them, does not exceed a specific value t. Also, another definition could be based on the edge connectivity of vertices. The edge connectivity of a pair of vertices in a graph is a minimum number of edges that should be removed to disconnect those two vertices. Vertices with large values of edge connectivity are more probable to be in the same

community. Local properties of subgraphs made a wide range of definitions for communities possible.

It is possible to define communities with considering graphs as a whole. These types of definitions are useful in case that communities can not be taken apart without significant change in the functionality of the system. These community definitions indirectly deliver communities by capturing some global properties of the graph. One proper class of these global definitions is based on the idea that a graph with community structure is different from a random graph. For example in a $G(n,p)$ random graph all the pairs of vertices are adjacent with the same probability p, therefore no community structure is expected. *Null* model of graph $\mathcal{G}$ is a random graph with some structural features similar to the graph $\mathcal{G}$. One good example of a null model consists of the randomized version of the original graph which expected degree of each vertex is equal to the degree of the corresponding vertex in the original graph. Null model provides a powerful tool to design community detection algorithms. Following, some community detection algorithms are described.

A. Quality Functions

Some community detection algorithms output a large number of partitions, which those partitions may not correspond to meaningful communities. In these cases a quantitative criterion could be helpful to assess the quality of partitions as a community structure. A quality function assign a number to each partition, that could be a measure to rank partitions based on their quality. One example is a quality function P which counts number of correctly interpreted pairs of vertices. For a partition $\mathcal{P}$ of graph $\mathcal{G}(V,E)$ we have

$$P(\mathcal{P}) = \frac{\big|(i,j) \in E, C_i = C_j\big| + \big|(i,j) \notin E, C_i \neq C_j\big|}{n(n-1)/2}. \tag{17}$$

Modularity of Newman and Girvan, a popular criterion for assessing quality of partitions, is based on the idea that a random graph is not expected to have a community structure. Existence of community structures can be revealed by comparison between the original graph and its null model. It is possible to write the following equation for modularity:

$$Q = \frac{1}{2m} \sum_{\forall i,j \in V} (A_{ij} - \frac{k_i k_j}{2m}) \delta(C_i, C_j). \tag{18}$$

A and m represent adjacency matrix and number of all the edges inside the graph, respectively. $\frac{k_i k_j}{2m}$ is the expected number of edges between vertices i and j in the null model which its degree distribution equals to degree distribution in the original graph. If i and j belong to the same cluster then $\delta(C_i, C_j)$ is equal to one and otherwise is zero. eq. (18) just considers pairs of vertices that belong to the same cluster so it is possible to rewrite it in the form

$$Q = \sum_{c=1}^{n_c} \left[\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right]. \tag{19}$$

where n_c is the number of clusters and l_c, d_c are the number of edges in the cluster c and sum of its degrees of the vertices respectively. Subgraph with positive value of modularity is supposed to have cluster structure. The subgraphs with higher values of modularity are more like a cluster, because in these cases number of internal edges are much more than expected number of edges in the null model of the graph.

Modularity is used as a quality function in many algorithms. Using modularity we can evaluate stability of partitions and compare them. Modularity is the best and the most widely used quality function by far. It is reasonable to assume that high values of modularity correspond to good partitions in the graph (not always). Optimization of modularity itself is a popular method for community detection in graphs. It has been proved that modularity optimization is an NP-complete problem. There are a lot of community detection algorithms that try to optimize the modularity function heuristically. For example, we can name algorithms based on the greedy techniques, simulated annealing, extermal optimization and etc.

Sometimes we deal with weighted graphs. Assume that W_{ij} represents weight of the edge between vertices i, j and sum of the weights of edges adjacent to the vertex i equals to d_i. Modularity can be generalized by the formula

$$Q_w = \frac{1}{2W} \sum_{\forall i,j \in V} (W_{ij} - \frac{d_i d_j}{2W}) \delta(C_i, C_j), \tag{20}$$

where W is the sum of weights of all the edges in the graph. This formula could be easily extended to the case of directed and weighted graphs.

B. Another Example

In this section, we describe *k-means clustering*, the most popular clustering algorithm, that divides items in to k different clusters. In k-means clustering algorithm for a set of points $X = \{x_1, x_2, \cdots x_n\}$, the cost function is

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - c_i\|^2 \tag{21}$$

The cost function should minimized to result in a good clustering. In eq. (21), $\|a-b\|$ represents the distance between given points a, b. S_i indicates the set of points points in the $i-$th cluster and c_i is its centroid. The algorithm works as follow:

- A set of n points are picked as a centroid in way that they are as far as possible from each other.
- Each vertex (point) from set X is assigned to its nearest centroid.
- New centroids are calculated for the clusters.
- After several rounds of iterations centroids of clusters become unchanged.

C. Spectral Algorithms

Another clustering algorithm that uses properties of eigenvalues of *unnormalized Laplacian* of graphs is called *spectral algorithm*. Assume that the weighted adjacency matrix of the graph $\mathcal{G}$ with n vertices is the matrix $W = (W_{ij})_{i,j \in \{1,2,\cdots n\}}$.

The matrix D is a diagonal matrix that its diagonal elements are equal to $d_1, d_2, \cdots d_n$. Unnormalized Laplacian of the graph is shown by

$$L = D - W. \tag{22}$$

It is easy to prove that L has n real eigenvalues of $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \lambda_n$. Also, it is not difficult to show that the number of connected components in the graph is equal to the multiplicity of the eigenvalue 0 in the unnormalized Laplacian matrix. For a graph with k connected components, eigenvectors of k smallest eigenvalues correspond to one of the graph's components. Algorithm 1 divides a graph into k different clusters based on the idea of relation between connected components in the graph and their corresponding eigenvalues.

Algorithm 1: Unnormalized spectral clustering

input : W, adjacency matrix of graph and number of clusters.

output: Clusters $C_1, C_2, \cdots, C_k$.

1. Compute the unnormalized Laplacian L. 2. Compute the eigenvectors correspond to the lowest k eigenvectors of L. 3. Build the $n \times k$ matrix V, whose columns are the k eigenvectors. 4. Correspond each column of V to a point and cluster those points (equivalently vertices of graph) with the $k-$means algorithm into clusters $C_1, \cdots, C_k$.

D. Detection of Dynamic Communities

Communities are fundamental units of all complex networks. Communities evolve over time, for example they have birth, death, merging and etc. Their structure and evolution are important in investigating properties of complex networks. In Figure 1 possible cases of community changes and evolutions are shown. Detecting communities in graphs are essential in order to investigating evolution of them. Analysis of dynamic communities is still an immature field of research because of difficulty of detecting all the communities in a network and unavailability of useful data on large real graphs.

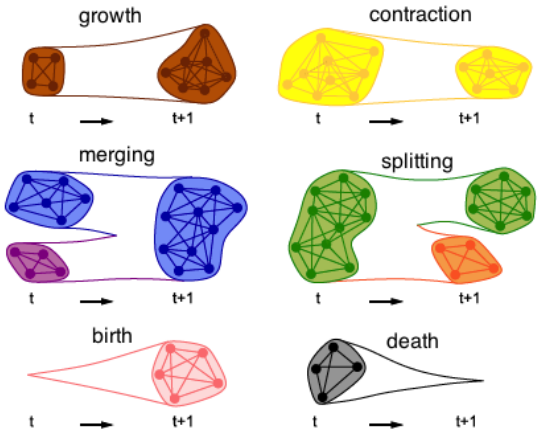

Fig. 1. Evolution of Communities in the Complex Networks [2]

A community evolution can be studied by the idea of defining a community over time. It is possible to define image of a community $\mathcal{C}(t+1)$ at time $t+1$ as the community $\mathcal{C}(t)$ with the largest overlap with $\mathcal{C}(t+1)$, among all the communities of the graph at time t. But we should take in account that sometimes by this definition actual evaluation of communities may be missed.

Since keeping track of communities in different time steps is a difficult task, it would be more easier to monitor community of a given vertex over time. One good measure to evaluate persistence of communities over time can be

$$a_i^t(\tau) = \frac{\left|\mathcal{C}_i(t) \bigcap \mathcal{C}_i(t+\tau)\right|}{\left|\mathcal{C}_i(t) \bigcup \mathcal{C}_i(t+\tau)\right|} \tag{23}$$

where $\mathcal{C}_i(t)$ and $\mathcal{C}_i(t+\tau)$ are communities of vertex i at times t, $t+\tau$, respectively. The more slowly $a_i^t(\tau)$ decays, we can say that the more strongly vertex i is connected to its community. Studying evolution of dynamic networks would be more accurate if previous informations of graph involved in the model. The evolutionary clustering is built on a unified framework in which clusters are specified based on the current structure of the graph and the knowledge of cluster structures at previous times.

IV. An Impossibility Theorem for Clustering [3]

The intuitive goal of clustering is to divide a set of points into different clusters which points in the same clusters are close together and points of different clusters are far apart. There are different definitions for clusters and clustering algorithms, i.e. there is no universally agreed definition. Variety of definitions made developing a unified framework for clustering algorithms difficult. Clustering algorithms are unified up to the level of intuitive goal of clusterings. However wide range of methods form information theoretic approach to combinatorial optimization and probabilistic models that are based on diverse underlying principles often result in quantitatively different results. Different starting points and criteria usually lead to different taxonomies of clustering algorithms

We give an insight to the difficulty of defining a unified frame work for clustering algorithms by stating and proving an impossibility theorem for them. For a set of three simple properties it is shown that there is no clustering function to satisfy them. Note that a clustering function is a functions f that takes a set S of n points and their pairwise distances as its input and returns those input points in several partitions. This impossibility theorem indicates natural inherent trade-offs in the clustering problem. Impossibility of this properties to be fulfilled by a clustering function hands in a way to categorize clustering functions based on the method they resolve this trade-off. In the following section we explain impossibility theorem in details.

A. Impossibility Theorem

Clustering is the assignment of a set of points $S = \{1, \cdots n\}$ into subsets, called clusters, so that points in the same cluster are similar in some sense. Between all pairs of data points

i and j a notion of distances, $d_{ij} \geq 0$, is defined. Distance function $d : S \times S \to \mathbb{R}$ is defined to satisfy the following conditions:

1) Symmetry. $D(i,j) = D(j,i)$.
2) Positivity. $D(i,j) \geq 0$ for all $i, i \in S$.

It is possible to study those distance functions that are metrics. A metric distance function in addition to above properties should satisfy the following conditions:

3) Triangle inequality. $D(i,j) \leq D(i,k) + D(k,j)$ for $\forall i,j,k \in S$.
4) Reflexivity. $D(i,j) = 0$ iff $i = j$.

A clustering function is a function f that takes a distance function d on the set S and returns a partition Γ of S. The sets in Γ will be called clusters of set S. In order to distinguish between "good" and "bad" clustering algorithms, it would be useful to require partitions to satisfy a set of basic generally and widely accepted properties. Here we define a set of three properties for clustering functions: *Scale-Invariance*, *Richness* and *Consistency*.

If d is a distance function, we write $\alpha \cdot d$ to denote the distance function in which the distance between i and j is $\alpha d(i;j)$.

Definition 1: **Scale-Invariance.** For any distance function d and any $\alpha > 0$, we have $f(d) = f(\alpha d)$.

Let Range(f) denotes the set of all partitions Γ such that $f(d) = \Gamma$ for some distance function d.

Definition 2: **Richness.** Range(f) is equal to the set of all possible partitions of S.

Let Γ be a partition of S, and d and d' two distance functions on S. We say that d' is a $\Gamma - transformation$ of d if

(a) For all $i,j \in S$ belonging to the same cluster of Γ, we have $d'(i,j) \leq d(i,j)$.
(b) For all $i,j \in S$ belonging to the different cluster of Γ, we have $d'(i,j) \geq d(i,j)$.

Definition 3: **Consistency.** Let d and d' be two distance functions. If $f(d) = \Gamma$, and d' is a $\Gamma-$transformation of d, then $f(d') = \Gamma$.

These three properties give a good formal relation between mathematical model and what we expect intuitively from a clustering function. Scale-Invariance says that it is good for a clustering function to cluster points independent of distances scales and just detect clusters based on the relative distances of points. Sometimes given pairwise distances among points, intuitive detection of clusters is a trivial task. Also, for a given partition of points, it is possible to define distances in a way (for example, assign small values to distances between points in the same partition and assign large values to distances between points of different partitions) that the corresponding intuitive clusters match to the given set of partitions. The goal of Richness property is to satisfy these motive. For clustering of points with a function f, which its input is a given set of pairwise distances, if distances between points in the same clusters decreased and distances between points of different clusters increased; it is obvious that we expect new output of cluster function with these new distances to be equal the output of f for the first set of distances. Consistency property is a good tool to model this observation.

In the following, first we show that there are clustering algorithms that satisfy each pairs of these three properties. Concept of *antichain* is defined and we show that the range of a clustering function with Scale-Invariance and Consistency properties is an antichain. This result contradicts with the Richness of clustering function. This will prove the so called impossibility theorem.

B. Single-linkage

Single-linkage starts with assigning each point to its own cluster. Distance between two clusters X and Y is defined as

$$d(X,Y) = \min_{x \in X, y \in Y} d(x,y).$$

Single-linkage procedure in each step finds the two clusters with the smallest distances and merges them into a one new cluster. Equivalently it is possible to construct a complete weighted graph with the set of points S as its vertices and putting weights of edges equal to distances between points. Edges are sorted non-decreasingly based on their weights. All edges are removed from the graph and one edge from the sorted list is added to the graph in each step. Nodes in each connected component of graph correspond to the points in the same cluster. It is possible to define various clustering functions by choosing different stopping condition for the single-linkage procedure. The first one is *$k-$cluster stopping condition* that procedure of cluster merging stops when the graph is consist of k connected component. *Distance-r stopping condition* only adds edges of weight at most r to the initial graph. Assume that ρ^* is the maximum weight of edges in the graph, i.e. $\rho^* = \max_{i,j \in S} d(i, yj)$. *Scale$-\alpha$ stopping condition* only adds edges of weight at most $\alpha\rho^*$.

Theorem 1: For any $k \geq 1$, and any $n \geq k$, single-linkage with the the $k-$cluster stopping condition satisfies Scale-Invariance and Consistency.

Proof:

- **Scale-Invariance.** If pairwise distances of points multiplied by α, order of adding edges to the graph in the single-linkage procedure does not change and the final clusters remain similar to the the first one.
- **Richness.** Output of clustering functions could not be equal to cases that number of clusters is different from k, because in this algorithm final number of clusters equal to k.
- **Consistency.** Set of picked edges by the single-linkage algorithm have smaller weights than those edges that have not been picked during the clustering procedure. If intra-cluster distances decreased and inter-cluster distances increased, first again the same set of edges are picked and we end-up to the similar set of k clusters.

■

Theorem 2: For any positive $\alpha < 1$, and any $n \geq 3$, single-linkage with the the scale$-\alpha$ stopping condition satisfies Scale-Invariance and Richness.

Theorem 3: For any $r > 0$, and any $n \geq 2$, single-linkage with the the distance$-r$ stopping condition satisfies Richness and Consistency.

Proof of theorems 2 and 3 are very similar to the proof of theorem 1 and we omit them here.

Definition 4: **Refinement of a partition.** A partition Γ' is a *refinement* of a partition Γ if for every set $C' \in \Gamma'$, there is a set $C \in \Gamma$ such that $C' \subset C$. The notation $\Gamma' \preceq \Gamma$, which means Γ' is refinement Γ indicates a partial order on the set of all the partitions.

Definition 5: **Antichain** is a subset of all the possible partitions which dose not contain two distinct partitions such that one of them is refinement of the other.

Theorem 4: If a clustering function f satisfies both Scale-Invariance and Consistency properties, then Range(f) is an antichain.

Proof: At first we define two new concepts.

Definition 6: For a partition Γ, we say that a distance function d, $(a, b) - conforms$ to Γ if, for all pairs of points i, j that belong to the same cluster of Γ, we have $d(i, j) \leq a$, while for all pairs of points i, j that belong to different clusters, we have $d(i, j) \geq b$.

Definition 7: A pair of positive real numbers (a, b) is $\Gamma - forcing$ if, for all distance functions d that $(a, b)-$conform to Γ, we have $f(d) = \Gamma$.

Theorem 5: Assume that a f is a consistence clustering function. For all $\Gamma \in$ Range(f), There exist $a, b > 0 \in \mathbb{R}$ such that the pair (a, b) is $\Gamma-$forcing.

Proof: For a given $\Gamma \in$ Range(f) there excites a distance function d such that $f(d) = \Gamma$. We define values of a' and b' in the following way:

$$a' = \min\{d(i, j)|\ \forall i, j \text{ in the same cluster of } \Gamma\}$$

$$b' = \max\{d(i, j)|\ \forall i, j \text{ that are in differene clusters of } \Gamma\}$$

Two values a, b are picked such that $a \leq a'$ and $b \geq b'$. By Consistency of f, it is straightforward to conclude that (a, b) is $\Gamma-$forcing. ■

Now assume that f is Scale-Invariance in addition to the has Consistency property. By contradiction we show that Range(f) is antichain. Let Γ_0 and Γ_1 are two sets of possible clustering in the Range(f) and assume that Γ_0 is a refinement of Γ_1. Based on the theorem 5 there exist two pairs of parameters $(a_0, b_0)(a_0 < b_0)$ and $(a_1, b_1)(a_1 < b_1)$ such that the first one is a Γ_0-forcing pair and the second is a Γ_1-forcing pair . Choose a_2 and ε such that $a_2 < a_1$ and $0 < \varepsilon < a_0 a_2 b_0^{-1}$. Distance function d is built based on the following properties:

- For all points i, j belong to the same cluster in Γ_0, which also are in the same cluster in Γ_1: $d(i, j) \leq \varepsilon$.
- For all points i, j belong to the same cluster in Γ_1 and no to the same cluster in Γ_0: $a_2 \leq d(i, j) \leq a_1$.
- For all points do not belong to the same cluster in Γ_1: $d_(i, j) \geq b_1$.

The distance function d, (a_1, b_1) conforms to Γ_1, so we have $f(d) = \Gamma_1$. Now new distance function $d' = \alpha \cdot d$ with the parameter $\alpha = b_0 a_2^{-1}$ is defined. We have assumed that d is Scale-Invariance and therefore $f(d') = \Gamma_1$. Because $\alpha\varepsilon < a_0$ and $\alpha a_2 \geq b_0$, thus $d', (a_0, b_0)-$conforms to Γ_0 and therefore $f(d') = \Gamma_0$. By the assumption $\Gamma_0 \neq \Gamma_1$, and this is a contradiction. ■

Theorem 6: For each $n \geq 2$, there is no clustering function f that satisfies Scale-Invariance, Richness, and Consistency.

Proof: In Theorem 4 we proved that Range(f) of a clustering function f with Scale-Invariance and Consistency properties is an antichain. For a set of $n \geq 2$ points, the collection of all partitions does not from an antichain. Therefore it is not possible for a clustering function to satisfy Scale-Invariance, Richness, and Consistency simultaneously. ■

V. Research Proposal

There is a quote from Marshall McLuhan, a media theorist, that says "We shape our tools and thereafter our tools shape us". Recommendation and personalization shape information flow in networks. It is interesting to say that "Personalization could easily have a hand not only in who goes on a date with whom but in where they go and what they talk about". We want to design a model for explaining user-item interactions in recommender systems and to see how and why networks of users/items and their communities evolve over time as a result of recommendations. Netflix Prize winner algorithm is a good choice to be studied for investigating effect of recommendation systems. Another research question is to model the relation between user's interests and recommended items to him/her to find effects of them on each other. If recommender system receives accurate feedbacks from users and apply those feedbacks in a correct way, after a while all the items suggested to the users would be based on their preferences. Now the question is that what would happen if the recommender system receives weak feedbacks from the users. Community detection algorithms and clustering functions provide us powerful tools for answering all the above questions.

It is possible to define other interesting research problems. Analysing dynamic networks is an immature field of research and designing new methods to track evolution of communities over time is a good way to study them. In this report, three axioms for clustering functions were defined and we proved that there is no clustering function to satisfy all of them simultaneously. The question is what can we say about clustering functions if we have some relaxations over the mentioned axioms?

References

[1] Y. Koren. The BellKor Solution to the Netflix Grand Prize, 2009. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[2] S. Fortunato. Community detection in graphs. *Phys, Rep. 486*, pp. 75-174, 2010.

[3] J. Kleinberg. An impossibility theorem for clustering. *Proc. 2002 Conf. Advances in Neural Information Processing Systems, vol. 15*, pp. 463-470, 2002.

[4] Y. Koren. Collaborative Filtering with Temporal Dynamics. *Proc. 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD09)*, pp. 447-456, 2009.

[5] NP R. Salakhutdinov, A. Mnih and G. Hinton. Restricted Boltzmann Machines for Collaborative Filtering. *Proc. 24th Annual International Conference on Machine Learning*, pp. 791798, 2007.