Tractable Group Detection on Large Link Data Sets

Jeremy Kubica
Carnegie Mellon University
Robotics Institute
Pittsburgh, PA 15213
jkubica@ri.cmu.edu

Andrew Moore
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
awm@cs.cmu.edu

Jeff Schneider
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
schneide@cs.cmu.edu

Abstract

Discovering underlying structure from co-occurrence data is an important task in a variety of fields, including: insurance, intelligence, criminal investigation, epidemiology, human resources, and marketing. Previously Kubica et. al. *presented the group detection algorithm (GDA) - an algorithm for finding underlying groupings of entities from co-occurrence data. This algorithm is based on a probabilistic generative model and produces coherent groups that are consistent with prior knowledge. Unfortunately, the optimization used in GDA is slow, potentially making it infeasible for many large data sets. To this end, we present k-groups - an algorithm that uses an approach similar to that of k-means to significantly accelerate the discovery of groups while retaining GDA's probabilistic model. We compare the performance of GDA and k-groups on a variety of data, showing that k-groups' sacrifice in solution quality is significantly offset by its increase in speed.*

1 Introduction

Co-occurrence data is an increasingly important and abundant source of data in many fields. In one general form the input consists of a series of *links*. Each link is an unordered set of entities that have been joined by some event, co-occurrence, or relation. For example, in the co-authorship domain each paper can define a link containing its authors as entities. It is important to appreciate that our definition of a link may vary from other usages, such as a link on a webpage. We do not assume any directionality or restrict links to contain a fixed number of entities.

One important task is the identification of underlying structure buried within large amounts of noisy link data. Below we examine the group detection algorithm (GDA), an algorithm that attempts to find underlying groups of entities [5]. GDA is based on a probabilistic generative model, in the form of a Bayesian network, that assumes that links are the result of underlying groups. GDA has been found to produce groups that are consistent with prior knowledge. Unfortunately GDA makes use of heuristic optimization techniques that may be slow, making it potentially infeasible for many real world data sets.

We propose k-groups, an algorithm that uses a probabilistic model similar to GDA but uses localized updates to improve both speed and convergence properties. We compare its performance to that of GDA on a variety of data. We show that k-groups' sacrifice in solution quality is offset by its increase in speed. This trade-off makes group detection tractable on significantly larger data sets.

2 The Group Detection Algorithm (GDA)

The group detection algorithm (GDA) uses maximum likelihood estimation to find groupings of entities from two input data sets [5]. The first is demographics data (*DD*), which contains the N_P entities and their demographic information. The word "demographic" should not be interpreted too narrowly as it can include any information about the entities. For example in the co-authorship domain, demographics may include an author's title or affiliation. We denote a single entity as e_i and the set of all entities as ξ. The second data set is the link data (*LD*), which consists of a listing of N_L separate links. We denote individual links as sets $L_i \subseteq \xi$ containing $|L_i|$ entities. From these data sets GDA learns K groups, each denoted $g_i \subseteq \xi$.

GDA assumes a probabilistic generative model in the form of a Bayesian network with five components. This model defines a recipe for data generation. Initially K groups are created from the entities' demographics data (*DD*) and an underlying demographics model (*DM*). Each entity/group membership is considered with the *DM* indicating the probability the entity is a member of the group given its demographics. From the underlying groups N_L separate links are generated and stored in a link data set (*LD*). The exact model for link generation is described below. The link model (*LM*) controls the amount of noise added to links of various types.

The goal is to learn the *LM*, *DM*, and groupings from the two input data sets (*DD* and *LD*). The primary component of interest is the chart (*CH*), which indicates entities' memberships in groups. Unlike traditional clustering models, an entity can simultaneously be a full member of several groups. Heuristic optimization is used to find the chart that produces the highest loglikelihood. As shown in [5] the Bayesian network structure allows the likelihood to be factored into two main components for optimization: the probability of the groups given the demographics $P(CH|DD,DM)$ and the probability of the links given the groups $P(LD|LM,CH)$. For the purposes of the k-groups algorithm we focus on optimizing $P(LD|LM,CH)$ and thus on the components of the model that relate to link generation from the groups.

The model assumes that links are generated individually by choosing a group g and uniformly sampling members from it. The amount of noise in a link is determined by the LM. We say that a link L contains noise if it was generated by some group g and there is some entity $e \in L$ such that $e \notin g$. During link generation, this corresponds to choosing each entity in the link directly from g or with some probability P_R as noise (from $\xi - g$). We can then break down the link size as $|L| = M_R + M_G$, where M_G is the number of entities in the link that are also in the generating group g and M_R is the number of entities that are noise (i.e. not in g).

In addition, with some probability P_I the link is completely random and all of its entities are chosen from ξ. In this case we define the *world group*, $G_W = \{e_j : 1 \leq j \leq N_P\}$, as the link's generating group. We then define $\Omega = \{g_1, \ldots, g_K, G_W\}$ as the set of all possible link generators. In other words, a link can be created by any of the K groups or be completely random (created by G_W).

Assuming the links are i.i.d. and the priors for each group are equal:

$$\begin{aligned} \log(P(LD|LM,CH)) &= \sum_{i=1}^{N_L} \log P(L_i|LM,CH) \\ &= \sum_{i=1}^{N_L} \left[\sum_{g \in \Omega} P(L,g|LM) \right] \end{aligned} \tag{1}$$

where

$$P(L,g|LM) = \begin{cases} \frac{P_I}{\binom{N_P}{|L|}} & if \;\; g = G_W \\ \frac{(1-P_I)}{K} P(L|LM,g) & if \;\; g \neq G_W \end{cases} \tag{2}$$

and (if $g \neq G_W$)

$$P(L|g,LM) = \left(\frac{(P_R)^{M_R}(1-P_R)^{M_G}\binom{M_G+M_R}{M_R}}{\binom{|g|}{M_G}\binom{N_P-|g|}{M_R}} \right) \tag{3}$$

An important approximation to the above equations is (4), which creates the concept of a group "owning" a link.

$$\log(P(L|LM,CH)) \approx \log\left(\max_{g \in \Omega} P(L|g,LM) \right) \tag{4}$$

3 The K-groups Algorithm

The k-groups algorithm focuses on the task of learning the underlying groups directly from the link data. That is to say, it optimizes $P(LD|LM,CH)$ and does not consider the demographics information in the inner loop.

Intuitively k-groups optimizes $P(LD|LM,CH)$ in a fashion similar to that of the k-means algorithm. Since the groups formed by k-groups are not disjoint, the k-means approach must be adapted by noting that the max approximation forces groups to "own" links. If we let LG_g be the set of links owned by a group $g \in \Omega$:

$$\begin{aligned} \log(P(LD|LM,CH)) &= \sum_{g \in \Omega} \log(P(LG_g, g|LM)) \\ &= \sum_{g \in \Omega} \sum_{L \in LG_g} \log(P(L,g|LM)) \end{aligned} \tag{5}$$

Similar to the k-means algorithms, k-groups can alternate (until convergence) between two greedy steps:

1. For each link, determine which group owns it.
2. For each group g, determine which entities form g so as to optimize $P(LG_g|g,LM)$.

3.1 Determining Group Ownerships

The first step of the k-groups algorithm is to determine which links each group owns. This can be done quickly with a single linear scan through the links using the approximation in (4):

$$LG_g = \{L : g = \underset{g' \in \Omega}{argmax} P(L,g'|LM)\} \tag{6}$$

3.2 Updating the Groups

The second step of k-groups is to find the "optimal" groups given their links. Since we are considering only the links owned by the current group we wish to optimize $P(LG_g,g|LM)$ for each g. We can do this locally by using (3) and (5) to examine the effect of adding or removing a single entity. Let $\Delta_g^A(e)$ and $\Delta_g^R(e)$ denote the change in $\log P(LG_g,g|LM)$ if we add entity e to group g and remove entity e from group g respectively. Then by algebra:

$$\Delta_g^A(e) = \begin{cases} \sum_{L \in LG_g} \left[\log\left(\frac{(|g|-M_G+1)(N-|g|)}{(N-|g|-M_R)(|g|+1)} \right) \right] + & \\ \sum_{L \in LG_g : e \in L} \left[\log\left(\frac{(1-P_R)(N-|g|-M_R)}{P_R(|g|-M_G+1)} \right) \right] & e \notin g \\ 0 & e \in g \end{cases} \quad (7)$$

with a similar result for $\Delta_g^R(e)$.

Using these observations we can define a second, inner greedy procedure to optimize $P(LG_g|g, LM)$:

1. For each $e \in LG_g$ calculate $\Delta_g^A(e)$ and $\Delta_g^R(e)$.
2. If there exists some entity e such that $\Delta_g^A(e) > 0$ (or $\Delta_g^R(e) > 0$) then add (or remove) the entity that would lead to the greatest improvement.
3. If $\Delta_g^A(e) \leq 0$ and $\Delta_g^R(e) \leq 0 \ \forall e$, terminate.

This procedure greedily adds/removes entities until it no longer results in an improvement. This approach has two major computational advantages. First, we do not have to recalculate $P(LG_g, g|LM)$ for each change that we wish to evaluate. This allows us to rapidly try each entity exhaustively. Second, k-groups only needs to consider a subset of links and a subset of entities when updating a group. It is important to appreciate that this search is localized and therefore may not find groups that optimize the overall likelihood of the data.

3.3 Convergence and Local Minima

One important advantage of k-groups is that it provably converges to a local optimum in a finite number of steps [4]. Despite this guarantee, k-groups may get trapped in a local optimum. We use two simultaneous strategies for "getting unstuck": a step similar to Split-Merge EM [7] and a step that adds a small amount of noise to the solution. Both serve to perturb the solution before k-groups is rerun, making each convergence a single iteration of the overall k-groups algorithm.

4 Comparison with GDA

The key advantage of k-groups is that it finds "better" solutions faster than GDA. We examined this improvement on a variety of real-world and synthetic data. Due to space considerations the experiments on the synthetic data sets are omitted. Results and analysis can be found in [4].

The tests on the real world data sets, summarized in Table 1, consisted of tracking the optimization performance (loglikelihood) versus the time for both algorithms. The data sets consist of links built from co-authorship (lab, citeseer, and institute), co-occurrence (manual and drinks), advisor/advisee relations (institute) and common research interests (institute). The group sizes and parameters were heuristically chosen, but constant across algorithms. Both algorithms were run multiple times on each data set.

DATA SET	N_P	N_L	K	# RUNS
LAB	115	94	20	24
INSTITUTE	456	1738	100	24
DRINKS	136	5325	50	24
MANUAL	4088	5581	25	8
CITESEER	104801	181395	50	1

Table 1. Summaries of the data sets.

DATA SET	K-GROUPS 1 ITR. LL	K-GROUPS TIME	GDA TIME	TIMES SPEEDUP
LAB	-726	0.13	145	1163.2
INSTITUTE	-18847	16.00	667	41.7
DRINKS	-42664	23.25	349	15.0
MANUAL	-86509	9.38	5033	536.9
CITESEER	-4763400	1319.00	N/A	N/A

Table 2. The average loglikelihood after one iteration of k-groups and the average time (in seconds) to reach that loglikelihood.

Figure 1 shows the results of some of the runs. At each time step the plot indicates the mean loglikelihood and the bounds of the 95% confidence interval on performance. No confidence intervals were included on the Citeseer results. As the results illustrate, k-groups can offer a significant speedup. As the size of the data sets increase (number of links and entities) this speedup can become more pronounced and also more important. For example, on the Citeseer data set k-groups was able to converge to a local minima in under 22 minutes that was better than any solution found by GDA within 24 hours.

The above results lead to another natural question: "How good is the first local minima found by k-groups?" For each data set we examined the average loglikelihood after a single iteration of k-groups (convergence to the first local minima) and the average time iteration took. This was then compared to the average time it took GDA to reach this loglikelihood. The results, shown in Table 2, indicate that a single iteration often performs relatively well and is *much* faster than finding a equally good solution using GDA.

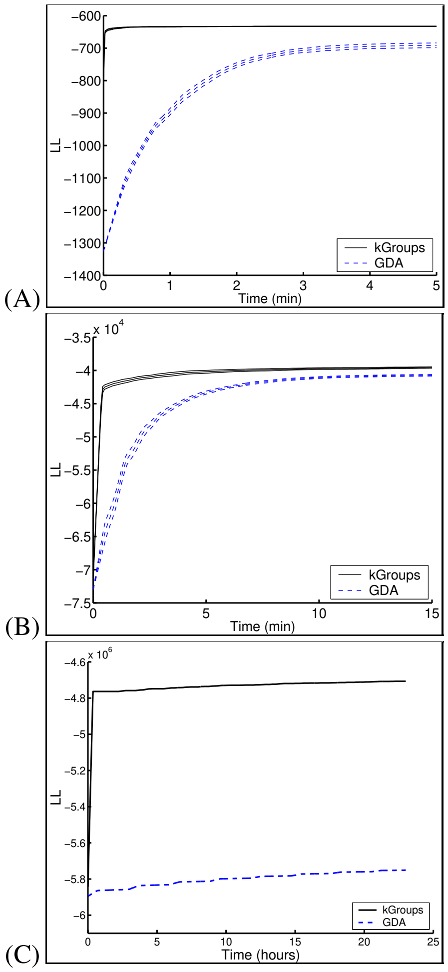

Figure 1. Loglikelihood versus time for (A) Lab, (B) Drinks, and (C) Citeseer data.

5 Related Work

The k-groups algorithm is an improvement of the GDA algorithm [5]. In addition there are a variety of similar algorithms that extract different types of structure from link data. Descriptions of these algorithms and how they compare to the GDA algorithm can be found in [4].

K-groups is similar to, and inspired by, the k-means algorithm [2]. Both algorithms use hard clustering for group membership and use only members to update the group. The two algorithms differ significantly on their domains and optimization criteria. The use of a split/merge operation was proposed by Ueda *et. al.* for use in EM optimization of mixture models [7].

When rephrased as the problem of assigning link ownerships to groups, our work becomes similar to approaches that cluster the links themselves [1, 3, 6]. There are several key differences between these approaches and our own. First, we are using a relatively novel underlying generative model. Second, we are primarily interested in the resulting "clusters" of entities, which we restrict to be hard assignments. Finally, k-groups uses a novel greedy approach that is designed for both this model and the hard, but nonexclusive, assignments of entities to groups.

6 Conclusions

Above we presented k-groups, an algorithm that uses localized updates to improve both speed and convergence properties while still using GDA's probabilistic model. We motivating the derivation of the algorithm using the same properties as k-means and compared k-groups' performance to that of GDA on several data sets.

There are several remaining questions that we plan to investigate in future research. The first is to learn K while learning the underlying groups by using a measure such as AIC or BIC. The second is to incorporate the demographics into the localized update steps. Finally, it is possible that by combining (or alternating) the GDA and k-groups optimization steps, the resulting optimization could ultimately find even better solutions quickly.

Acknowledgements

Jeremy Kubica is supported by a grant from the Fannie and John Hertz Foundation. This research is supported by DARPA under award number F30602-01-2-0569. The authors would like to thank Alex Gray for his helpful comments and Steve Lawrence for providing the Citeseer data.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.

[2] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Communications and Information Theory. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[3] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[4] J. Kubica, A. Moore, and J. Schneider. K-groups: Tractable group detection on large link data sets. In *CMU Tech. Report 03-32*, 2003.

[5] J. Kubica, A. Moore, J. Schneider, and Y. Yang. Stochastic link and group detection. In *AAAI*, pages 798–804. ACM Press, Jul 2002.

[6] K. Nigam, A. K. McCallum, S. Thrun, and T. M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[7] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. SMEM algorithm for mixture models. *Neural Computation*, 12(9):2109–2128, 2000.