Clustering of Correlated Documents into Designated Number of Clusters: A Practical Approach

Ali Cevahir[1]

[1] Bilkent University, Department of Computer Engineering,
06800 Ankara, Turkey
acevahir@cs.bilkent.edu.tr
http://www.cs.bilkent.edu.tr/~acevahir/cs550

Abstract. We consider a complete graph to cluster the vertices into *k*-clusters, where each edge of the graph is labeled either as "+" or "–". "+" denotes that vertices incident to the edge are mutually related and "–" edge denotes that the vertices incident to the edge are mutually unrelated. The goal of the clustering is to place vertices into *k* clusters, where documents are clustered with maximally related items. That is, clustering should maximize the agreements ("+" edges inside the clusters and "–" edges between the clusters), or equivalently minimizes the disagreements ("–" edges inside the clusters and "+" edges between the clusters). We give a simple algorithm for the maximizing the agreements and test the success of the algorithm. We compare approach with Bansal et. al's approach proposed in [1]., and conclude that complicated algorithms that have exponential run time are not practical.

1 Introduction

In this paper, we examine clustering of *n* correlated documents into *k* partitions. Problem is represented by a complete graph. Each vertex represents a document and each edge represents a relation between documents. Each pair of documents are either related or not related. Related documents are connected with a "+" edge, and unrelated documents are connected with a "–" edge. The graph is a simple graph, which means if a document (will be called as "a vertex" from now on) A is related with B, then B is related with A. The relation data may be learned from the past experiments, where we do not deal how the relation information is learned, in this work.

The main objective of the clustering problem we study is finding an appropriate *k*-partitioning of a complete graph where maximum number of agreements is achieved. We mean by agreement that the positive edges inside clusters and negative edges across clusters. Equivalently, the problem can be defined as minimizing disagreements: number of "–" edges within the clusters and number of "+" edges between the clusters. Maximizing agreements and minimizing disagreements are equivalent problems; however they often require different points of views. In this work, we mainly deal with maximization problem.

The problem of correlation clustering is introduced in [1]. They explain that the problem formulation is motivated from the set of clustering problems at Whizbang

Labs ([4]), in which learning algorithms were trained with the help of various clustering tasks. One example application of the correlation clustering is clustering different names that address to the same identity. Think of a case of researchers. The same person might appear in the set multiple times with different affiliations or one with middle names.

Note that if a perfect clustering exists for a graph, i.e., if all edges within the clusters are "+" and all others (the ones between clusters) are "–", then the problem becomes trivial. All we have to do is to delete all negative edges and choose remaining complete graphs as clusters. This is called "naïve algorithm" in [3]. However it is not always possible to have perfect clusters. If we have triangles with two positive edges and one negative edge, it is impossible to cluster this data correctly, because all ways of clustering this triangle are erroneous. This data is erroneous, actually. Think of the nodes of such triangles and let them be A, B and C. Without loss of generality, let the edges AB and BC are positive and AC is negative. It means that A and B, and B and C are related, however A and C are not related, which is a wrong hypothesis in the context of our problem. Our problem is to find best representation with a limited knowledge, as in the case of agnostic learning. The problem is proved to be NP-hard by reduction from triangles GT11 [1].

One reason for data to be erroneous is the noise in data. When noise is small, polynomial time algorithms work well for clustering. However worst case analysis of this problem is much harder. In [1], worst case analysis for similar clustering problem is done. However, in [1], the number of clusters is not known, which makes problem a bit harder. In this work, we will show that their exponential time algorithm for worst case is not usable in practice and unnecessary for most of the time since simpler polynomial time algorithms achieve well.

Different approaches are applicable for the problem. For example, min-sum clustering of Schulman [6], min-max clustering of Hochbaum and Shymons [5] and *k*-median clustering of Charikar [2] may be applicable. However, it seems that the best possible approach to this problem is a linear programming approach, as in the case of Swamy's work [7], because the problem is a maximization (or minimization) problem where we have a well defined structure. However, this work addresses to the problem in the view of machine learning context. We introduce an unsupervised learning algorithm where we have no past information about clustering and each node finds its cluster with the help of previously clustered vertices in the graph.

We propose a polynomial time algorithm for this problem and try to improve the success of the algorithm, with a constant factor run-time penalty. However when we compare the results of the initial algorithm and the extended algorithm, we see that extension is not always successful, and learning with initially fewer knowledge is sometimes better. Also we discuss another extension of the algorithm, so that it can handle the correlation problem for unknown number of clusters.

The problem may be extended to real valued edges. We extend our algorithm to real valued edges and see that it performs with reasonable success in this case, too.

We organize the paper as follows. In Section 2, we explain our polynomial time algorithm. In Section 3, we present the results of the algorithm for both small noised data and random data. We discuss possible extensions of our algorithm in Section 4. In Section 5, we test the algorithm for real valued edge weights. In Section 6, we conclude.

2 The Polynomial Time Approximation Algorithm

In [1], two algorithms are represented for the correlation clustering problem. One of the algorithms is for minimizing the disagreements and the other is for maximizing agreements. Algorithm which minimizes disagreements runs with $O(n^3)$ time and guarantees to output a clustering with a constant factor approximation, where the number of mistakes (negative edges inside clusters and positive edges between clusters) made by the algorithm is less than a constant factor times the number of mistakes in optimal clustering. However, the constant factor for this problem is greater than 5, which is not a good approximation. Good approximations can be obtained by their second algorithm; however, its run time is exponential in $1/\varepsilon$, where ε is the approximation ratio. Latter algorithm is based on random sampling. They choose a small subset in whole set of vertices of the graph keeping in mind that its character is similar to the whole graph. In the algorithm they enumerate all partitions of this small set. However, even if the set is too small, there are many possible partitions of this set. For example, for a set of size 10, it has more than 115,000 possible partitions. To make their algorithm usable, one should select very small subset of vertex set, but in this case it is doubtful that very small sized subsets really characterize the whole set.

While examining such complex algorithms, one may want to know how well we can do with simpler polynomial time algorithms. Do we really need exponential time algorithms to reasonably classify a given set? As it will be seen in the later sections of this paper, our algorithm produces reasonably well clusters in polynomial time for most practical applications.

The algorithm starts with minimum knowledge about the graph. First, it randomly chooses k vertices to place each in a different cluster. Then, for each other vertices in the vertex set, we use current clustering information. Let we have clusters $C_1, C_2, \ldots, C_k$ during a point of execution, and we are to check for vertex v. We try to place v to a cluster C_i where the agreements are maximized. Without loss of generality, let C_1 is the best choice for such clusters. For the next iteration, we do the same check for one of the other vertices in the vertex set, say w, and clusters $C_1 \cup \{v\}, C_2, \ldots, C_k$.

The Algorithm

1. Let $G(V, E)$ be a complete graph to be clustered with for all edges e_i in E, weight of e_i is either -1 or 1.
2. Select k vertices in V randomly[1]. Place each vertex to a different cluster. Let A be the set which includes all vertices in V except the ones that are clustered.
3. Pick a vertex v in A randomly and do:
    a. Let we have clusters $C_1, C_2, \ldots, C_k$ formed until now by V-A.
    b. For each cluster, find agreements of v to V-A, when v is placed in the selected cluster. If C_i is the best match with v, $C_i = C_i \cup \{v\}$
    c. $A = A - \{v\}$
4. Repeat until A is empty

[1] To increase the success rate for small number of noised data, initially clustered vertices might be chosen as they have negative edges between them.

It should be easily seen that the run time of the algorithm is $O(n^3)$, which is very good for correlation clustering problems, because most of the algorithms proposed for correlation clustering is polynomial. The success of the algorithm for correctly clustering data is discussed in next section.

3 Analysis of Clustering Success for Noised and Random Data

Recall we explained in which conditions clustering problems come up. There might be a perfect clustering initially, and later data can be distorted by a ratio. This is called noise in data. In case of small noise, the problem is not very difficult to handle. Another possible clustering problem is that there is a random graph where positive and negative edges are uniformly distributed. This problem is harder to solve in often cases. We need exponential time algorithms for this type of problems. However, in most cases we do not need to deal with worst cases (the latter case).

We test the polynomial time approximation algorithm for both types of the problems and figure out that the algorithm returns nearly correct clustering for random noise. For random graphs also, the algorithm returns better clustering than randomly clustering the data, however since we do not know the mistakes in optimal clustering, we cannot compare our results.

The results for the tests of clustering with random noise are tabulated in *table 1* and *table 2*. While testing, we start with an initially perfect clustering, and add some noise to this data. Since noise is small (smaller than 0.1), we may assume that, the optimal clustering of noised data is same as the initial perfect clustering. We let our algorithm find the optimal clustering in noised data and compare our results with the ratio of noise.

Table 1. The ratio of approximations of the algorithm for small noised data. The results are for $k = 5$ and $n = 500$. For noise of ratio > 0.1, results may not be accurate. Smaller ratio means better approximations.

Ratio of noise	**Average ratio of approximation (%)**
0	Data is correctly classified
0.01	41.56
0.02	27.26
0.03	19.17
0.04	15.17
0.05	12.93
0.06	11.01
0.07	10.89
0.08	9.12
0.09	8.67
0.1	7.65
0.15	6.9
0.2	6.35

Table 2. The ratio of approximations of the algorithm for small noised data. The results are for $k = 20$ and $n = 500$. For noise of ratio > 0.1, results may not be accurate.

Ratio of noise	Average ratio of approximation (%)
0	Data is correctly classified
0.01	20.7
0.02	16.33
0.03	18.16
0.04	17.11
0.05	14.68
0.06	14.85
0.07	15.59
0.08	13.98
0.09	14.61
0.1	14.73
0.15	14.47
0.2	12.59

Here, we have to explain the terminology we used in the tables. *Mistake* of a clustering refers to negative edges inside the clusters and positive edges between the clusters. The *ratio of approximation* refers to ratio of extra mistakes done by a clustering to the mistakes of optimal clustering. Let m_{opt} be the number of mistakes in optimal clustering, and m_c is the number of mistakes of a clustering C. Then the ratio of the approximation C is $(m_c - m_{opt})/m_{opt}$. Let *err* be the ratio of erroneous edges in clustering to $|E|$. If the noise in data is known and it is sufficiently small, then we can calculate ratio of the approximation by (err – noise)/noise.

One of the most important problems for random small noise is, when the initial vertices are most probably misclassified. This misclassification affects other vertices to be misclassified for several iterations. However, after some number of iterations, algorithm classifies data more accurately.

For randomly distributed data, which is not a distortion of a perfect clustering, unfortunately we have no idea about the number of mistakes in optimal clustering. However, we may compare the performance of the algorithm against random clustering. Let S_{random} and S_C be the number of correctly clustered edges in random clustering and clustering C, respectively. An edge is correctly clustered if it is positive and it is within a cluster, or it is negative and it is an across edge between cluster. Now, we may define the success of C with respect to random clustering by calculating $(S_C - S_{random})/ S_{random}$. Tables 3 and 4 depicts the success of the algorithm with respect to random partitioning for graphs with 40%, 50% and 60% average number of positive edges.

The success rates of random data may seem to be small; however the optimal clustering cannot do much better, because there should be a constant rate for mistakes in optimal clustering since data is completely random. An analysis for the average number of mistakes in optimal clustering of randomly distributed data would help us to analyze the success rate of our algorithm, but unfortunately it is missing for this work.

Table 3. The success of the algorithm over random clustering for randomly distributed data. The results are for $k = 7$ and $n = 500$. Greater ratio means better approximations.

Average ratio of "+" edges in data (%)	Average success wrt. random clustering(%)
40	9.5
50	6.3
60	24.42

Table 4. The success of the algorithm over random clustering for randomly distributed data. The results are for $k = 15$ and $n = 500$.

Average ratio of "+" edges in data (%)	Average success wrt. random clustering(%)
40	10.87
50	5.81
60	17.85

4 Possible Extensions and Improvements for the Algorithm

As noted before, the initial clustering affects the performance of the algorithm. If the first iterations of the algorithm results with good clustering, then it is more probable that the optimal clustering is achieved.

Let us think of a possible extension. For initial clustering phase (step 2), place 3 vertices to each cluster, where each triplet should be in the same cluster (they have only positive edges between them). In opposition to our expectations, this modification degrades the performance of the algorithm. For example, for k=7, n=500, and average rate of positive edges in the data is 50%, the success rate of the modified algorithm with respect to random clustering is 5.65%, where original algorithm achieves a success rate of 6.3%. Sometimes minimum initial knowledge makes unsupervised learning algorithms perform better. Algorithms sometimes learn better when they learn themselves. If more accurate clustering is desired, extra passes may be applied to make polynomial time refinements after the termination of the algorithm.

The algorithm may be modified so that it handles the case when k (number of clusters) is unknown. To achieve this improvement, we are going to modify step 2 of the algorithm again. We are going to use an idea which is similar to the polynomial time approximation scheme of [1]. We pick a small, random subset of G. We check for all possible partitions of this subset and choose the best possible partitioning as initial clustering. The inspiration behind this idea is most possibly, the character of the subset is similar to G and each cluster in initial clustering is a subset of clusters in optimal clustering. This algorithm is faster than the one which is presented in [1]; however we still have the problem of enumerating all partitions of the initial subset. The initial subset should not be very small since very small subsets are doubtful for characterizing the whole set. The modified algorithm seems to work better than the algorithm presented in [1], since it learns about clustering with minimum initial knowledge; but unfortunately we lack of the test results to make a comparison.

5 Real Value Weighted Graphs

The algorithm is also applicable to weighted graphs, where the weights of the edges range between real values in [-1, 1]. All we have to do for applying the algorithm to the weighted graphs is to modify the scoring (step 3.b) to real valued scores. Another possible modification of the algorithm for real valued graphs is to assume all positive edges have weights 1 and all negative edges have weights -1. In this case, it is proven in [1] that if the original algorithm works with ρ-approximation, then it is guaranteed that modified algorithm works with $(2\rho + 1)$-approximation. However, we choose the first modification for real valued weight case, since it achieves better success than the worst case and our structure is suitable for both modifications.

Tables 5, 6 and 7 demonstrate the success of the algorithm for weighted graphs. We add random noise to the data such that if an edge's sign is negative, where we want to add noise, we flip its weight to a positive value randomly. We do the opposite for positive edges. Definitions for the success rates are similar to discrete-weighted case.

From the results obtained by the tests, it is clear that algorithm performs little worse for real valued weights. It is natural, because real value-weighted graphs are more difficult to study than the graphs with discrete value-weighted edges. Note that the accuracy of the tests for real value-weighted graphs is more uncertain, since the objective for this case is to maximize the total edge weights in the clusters and minimize the total edge weights between clusters. It is not to maximize the positive edges inside the clusters and negative edges between the clusters as in the former case (the original algorithm).

Table 5. The ratio of approximations of the algorithm for small noised, real value weighted data. The results are for $k = 5$ and $n = 500$. For noise of ratio > 0.1, results may not be accurate.

Ratio of noise	**Average ratio of approximation (%)**
0	Classifies 98,4% correctly
0.01	64.2
0.02	52
0.03	26.7
0.04	25.2
0.05	15.6
0.06	10.9
0.07	10.2
0.08	14.3
0.09	11.1
0.1	8.3
0.15	15.5
0.2	4.9

Table 6. The ratio of approximations of the algorithm for small noised, real value weighted data. The results are for $k = 20$ and $n = 500$. For noise of ratio > 0.1, results may not be accurate.

Ratio of noise	Average ratio of approximation (%)
0	Classifies 97.5% correctly
0.01	115.6
0.02	72.7
0.03	68.3
0.04	52.4
0.05	33.8
0.06	31.3
0.07	34.8
0.08	28.8
0.09	25.1
0.1	16.9
0.15	18.9
0.2	16

Table 7. The success of the algorithm over random clustering for randomly distributed data. Edge weights are random and uniformly distributed in [-1, 1]. $n = 500$.

Number of clusters (k)	Average success wrt. random clustering(%)
5	7.4
7	7.1
10	6.7
15	6.6
20	6.4

6 Conclusion

In this work, we have presented a polynomial time algorithm for correlation clustering problem with known number of clusters and tested the success of the problem. We discussed possible extensions and improvements of the algorithm and applied algorithm to graphs with real valued weights. Test results indicate that the algorithm produces sufficiently good clusters. The work reveals that we do not need to have exponential time algorithms for most practical applications, since the proposed polynomial time algorithm achieves a sufficiently good performance. However, application of the algorithm to the real value-weighted graphs may need some improvements. Also we lack of an analysis for the average number of mistakes in optimal clustering of randomly distributed data which would help us to analyze the success rate of our algorithm better.

References

1. Bansal, N., Blum A., Chawla S.: Correlation Clustering. Machine Learning, Vol. 56. Kluwer Academic Publishers. The Netherlands (2004) 89-113.
2. Charikar, M., Guha, S., Improved Combinatorial Algorithms for the Facility Location and *k*-Median Problems. In Proceedings of the 40th annual Symposium on Foundations of Computer Science (1999)
3. Cohen, W., Richman, J.: Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (2002)
4. Cohen, W., Richman, J.: Learning to Match and Cluster Entity Names. ACM SIGIR'01 Workshop on Mathematical/Formal Methods in IR (2001)
5. Hochbaum, D., Shmoys, D.: A Unified Approach to Approximation Algorithms for Bottleneck Problems. JACM, Vol. 33 (1986) 533-550
6. Schulman, L.: Clustering for Edge-Cost Minimization. In Proceedings of the 32th Annual ACM Symposium on Theory of Computing. (2002) 547-555
7. Swamy, C.: Correlation Clustering: Maximizing Agreements via Semidefinite Programming. In Proceedings of the Symposium on Discrete Algorithms. (2004)
8. Orlov, M.: Efficient Generation of Set Partitions. http://www.cs.bgu.ac.il/~orlovm/papers.html. Personal Web Page (2002).

Appendix: Source Code

The MATLAB source files for the proposed algorithm and extensions can be found at www.cs.bilkent.edu.tr/~acevahir/cs550. Please contact me for detailed information about the sources.