A new approach to clustering

Javed Aslam*, Alain Leblanc**, Clifford Stein***

Department of Computer Science
Dartmouth College
Hanover, NH 03755

Abstract. In this paper we present a new approach for clustering data for which the only information available is a similarity measure between every pair of elements. The algorithm makes no assumption about the size or number of clusters, or of any contraint in the similarity measure other that we expect it to be on average higher between elements belonging to the same cluster that for elements assigned to different clusters. The algorithm relies on very simple operations. The running time is dominated by matrix multiplication, and in some cases curve-fitting. We will present experimental results from various implementations of this method.

1 Introduction

The problem of clustering a data set into subsets, each containing similar data, has been very well studied. Several extensive studies on this topic have been written, including those of Everitt [4], Kaufman and Rousseeuw [6], and Mirkin [8]. In the typical clustering problem we are given a data set S on which a similarity measure ξ is defined, and we want to partition S into a collection of subsets $C_1 \dots C_k$ such 2 elements from the same subset C_i are more likely to have a higher similarity measure than if they were assigned to different subsets.

We can represent such data by an undirected weighted graph where the vertices represent the data and the weight of the edges is determined by ξ. Letting $\omega(a, b)$ denote the weight of edge ab, a graph can then be stored in a matrix M where $M_{ab} = \omega(a, b)$ (given that we label the edges $0, 1, 2 \dots n-1$ for a graph with n nodes), and M_a is the vector formed by row a.

In this paper we present a new approach for clustering a data set. Our objective is to develop an algorithm with the following properties:

1. The only assumption we make about the weight function is that $0 \leq \omega(a, b) \leq 1$ where a low value of $\omega(a, b)$ denotes a smaller similarity between a and b. (In general it should be possible to scale any positive weight function to be between 0 and 1.)

* This work partially supported by NSF Grants EIA-98-02068 and BCS-99-78116.

** Research supported by NSF Career Award CCR-9624828, a Dartmouth Fellowship, and NSF Grant EIA-98002068.

*** Research partially supported by NSF Career Award CCR-9624828, NSF Grant EIA-98-02068, a Dartmouth Fellowship, and an Alfred P. Sloane Foundation Fellowship.

2. The data can be noisy. That is, many pairs of elements that belong to different clusters can have a high similarity measure, and many pairs of elements belonging to the same cluster can have a low similarity measure.
3. We make no assumptions about the size and number of clusters. The algorithm will determine these values based on the data.
4. There is no human intervention between the time the program reads the data and the time the output is given.

Despite the extensive literature on clustering, we found very few references about clustering algorithms that respect the first three properties. The most interesting work that came to our attention was the paper by Ben-Dor et al [2] where they develop an algorithm to cluster gene expression data. However it is not clear how their algorithm performs when the average weights of clusters is not known beforehand, and how it will handle clusters with different average weight.

A typical example of an algorithm not respecting the first property is one which expects data to be mapped into a geometric space and uses the distance between two data points to compute the similarity. The same class of algorithms would not respect the second property, since geometric constraints often eliminate the possibility of having a lot of noise. If a and b are close together, and so are b and c, then a and c must also be near each other.

The third property is not followed by classical algorithms such as the k-means algorithm [4] or the self-organizing maps [7], unless the algorithm is part of a procedure which iterates on the number of clusters and uses an optimization function to determine which parameters produces the best clustering. These algorithms may not respect the fourth property either, since it is often the case that a human will be setting the parameters and select the best clustering based on his knowledge of the field.

Our approach In this paper we present an algorithm that satisfies the first three objectives. It generates a small set of possible solutions, of which very few (typically 3 or 4) are serious candidates. In a forthcoming paper we will present a method to extract an optimal clustering from these potential solutions, thereby satisfying condition 4.

Our approach is based on the simple observation that although $\omega(a,b)$ in itself may not be very reliable in deciding whether a and b should belong to the same cluster, the sets $\{\omega(a,x)|x \in S\}$ and $\{\omega(b,x)|x \in S\}$ can help in making this decision. More specifically, even if the process that computed the similarity between a and b produced a wrong measure, we still expect that for a sufficient number of elements x in S, $\omega(a,x)$ and $\omega(b,x)$ are an accurate representation of the similarity between a and x and between b and x. For instance, if $\omega(a,x)$ and $\omega(b,x)$ are close for most x's, then we can take this as an indication that a and b are likely to belong to the same cluster, regardless of $\omega(a,b)$.

We use a simple transformation to reweight the edges of a similarity graph based on this idea and which has the following property. Assume that the edges between vertices belonging to a same cluster (we will call them *intra-cluster* edges) are drawn from a distribution $D_1(\mu_1, \sigma_1)$ and the edges going across clusters (we will call them *inter-cluster* edges) are drawn from a distribution $D_2(\mu_2, \sigma_2)$ where D_1 and D_2 are arbitrary distributions. (In this paper μ and σ indicate the mean and standard deviation. All the operations on the edges that we describe can be applied to all the edges in M at once, or iteratively on each row of M. In the first case we say that we operate *globally*. In the second case we operate *by vertex*.) Then after the transformation the intra-cluster edges form a distribution which closely resembles a normal distribution $\tilde{N_1}(\tilde{\mu}_1, \tilde{\sigma}_1)$, and similarly the distribution of the inter-cluster edges closely resembles a normal $\tilde{N_2}(\tilde{\mu}_2, \tilde{\sigma}_2)$ where $\tilde{\mu}_1 > \tilde{\mu}_2$. In addition it is usually the case that $(\tilde{\mu}_1 - \tilde{\mu}_2)/\tilde{\sigma_2} > (\mu_1 - \mu_2)/\sigma_2$. Therefore, after the transformation, we have a better idea of how the edges are distributed even if we do not know the parameters of the distributions. Also the increase in the ratio of the difference of the means to the standard deviation of the inter-cluster edges means that the intra-cluster edges are relatively smaller. For instance there will be fewer inter-cluster edges with weight higher than the mean of the intra-cluster edges.

This leads to an algorithm where at its core is a series of iterations which can be summarized by the following three steps:

1. Reweight the edges resulting from the previous iteration.
2. Select a subset of edges to threshold (or set their weight to 0). By using our knowledge about the types of distributions that will be formed, we hope to threshold a much higher proportion of inter-cluster edges.
3. Set the edge weights to restart the next iteration.

The objective is that as the number of iterations increases there will be fewer and fewer inter-cluster edges remaining due to thresholding, until there are none left. At that point if there are enough intra-cluster edges left then we can identify the clusters by computing the connected components of the graph.

Although the idea behind each of these operations is simple, their exact implementation is not. We will present one alternative for step 1, three possibilities for step 2, and 4 scenarios for step 3. In addition, step 2 can be performed either by looking at all the edges at once, or working on the edges adjacent to each vertex individually.

2 The algorithm

We now present the algorithm. The pseudo-code is given in figure 1. The input is a similarity matrix M where $M_{ij} = \omega(i, j)$. The objective is that $M^{(t)}$ will contain more intra-cluster edges and fewer inter-cluster edges than $M^{(t-1)}$ does, until hopefully all the inter-cluster edges are gone. From $M^{(t)}$ we compute $C^{(t)}$, which is a set of clusters obtained by looking at $M^{(t)}$. We are interested in determining whether one of the $C^{(t)}$'s constitutes a valid clustering of the input set.

Step 1: Computing $M^{(t)}$ from $M^{(t-1)}$ The objective in computing $M^{(t)}$ is to be able to separate more easily the intra-cluster edges from the inter-cluster edges by reassigning new edge weights. We compute $M^{(t)}$ by setting

$$M_{ij}^{(t)} = \frac{M_i^{(t-1)} M_j^{(t-1)}}{(\|M_i^{(t-1)}\|)(\|M_j^{(t-1)}\|)}$$

Input: A similarity matrix $M^{(0)}$, and a number of iterations T.
Output: A collection of clusterings $C^{(t)}, t = 1..T$

for t = 1 to T:
- **step 1:** Compute $M^{(t)} = f(M^{(t-1)})$
- **step 2:** Select a set of edges $\bar{E}$ for thresholding.
  For each $uv \in \bar{E}$:
  $M^{(t)}_{uv} = 0 \ , \ M^{(t)}_{vu} = 0$
- **step 2.5:** Put in $C^{(t)}$ the connected components of $M^{(t)}$
- **step 3:** Assign weights in $M^{(t)}$ for the edges *not* in Θ.

Fig. 1. Pseudo-code for the algorithm

where M_i is the i'th row of M. $M^{(t)}_{ij}$ is the cosine of the angle between the vectors $M^{(t-1)}_i$ and $M^{(t-1)}_j$. It is 1 when the vectors are identical, and 0 when they are orthogonal. This measure is commonly used to evaluate the similarity of documents represented as vectors in the Information Retrieval community[1] [10].

Step 2: Thresholding the edges In section 4 we will see that at the end of step 1 we expect the edge weights in $M^{(t)}$ to form two normal distributions. The intra-cluster edges form a distribution $N(\tilde{\mu_1}, \tilde{\sigma_1})$, and the inter-cluster edges form a distribution $N(\tilde{\mu_2}, \tilde{\sigma_2})$. This is useful because if, for instance, $\tilde{\mu_1} > \tilde{\mu_2} + 3\tilde{\sigma_2}$, then by removing all the edges with weight lower than $\tilde{\mu_1}$ we remove most of the inter-cluster edges while preserving half the intra-cluster edges.

Although we do not know the values of μ_1 and μ_2, this information can still be used in order to select a set of edges $\bar{E}$ such that $\bar{E}$ is likely to contain a high proportion of inter-cluster edges. We will then set the weight of all the edges in $\bar{E}$ to 0. If, as a result, all or most of the inter-cluster edges are removed then we may be able to identify the clusters. Otherwise we use the resulting $M^{(t)}$ (after possibly some more modification as detailed in step 3) to compute $M^{(t+1)}$.

Given $M^{(t)}$ obtained after step 1 at the t'th iteration, we have devised three strategies to select which ones are to be put in $\bar{E}$. They are the *histogram* method, the *curve-fitting* method and the *probabilistic* method. The first two methods select a fixed thresholding value π and remove all the edges with weight smaller than π. In the probabilistic method every edge can be removed based on some probability function. We now describe each method.

Histogram In the histogram method we attempt to draw the curve of the distribution of the edges and see if we can distinguish between the two curves. When $(\tilde{\mu}_1 - \tilde{\mu}_2)/\tilde{\sigma_2}$ is large then it may be possible to identify the peaks formed by the intra-cluster and the inter-cluster edge weight distributions because there will be relatively few intra-cluster edges around $\tilde{\mu}_2$, and similarly few inter-cluster edges around $\tilde{\mu}_1$.
Let E be the set of edges under consideration, and let μ, σ be their mean and standard deviation. We draw the curve by building a histogram by assigning the edge weights to a predefined number m of buckets $b_0, b_1, \ldots b_{m-1}$ so that there are enough buckets to reasonably approximate the distribution, but not so many that each bucket receives just a few edges.

Let $n(b_i)$ be the number of edges in b_i. We can distinguish the two curves if there are three values u, v, w such that $n(b_i)$ is mostly increasing from b_0 until b_u, then mostly decreasing until b_v, then increasing again until b_w, and finally decreasing again until b_{m-1} with $n(b_v)$ significantly smaller than $n(b_u)$ and $n(b_w)$. Here we say "mostly" increasing and decreasing to allow for some small increase or decrease between the summit at u and w and the valley at v.
The exact values that we use we derived empirically. So far we have used a fixed number of buckets, and consider a b_i a summit only if $n(b_i)$ is greater than a certain fraction of the edges, and similarly b_j is a valley only when it is between two summit and $n(b_j)$ is smaller than a given fraction of the edges. When we can not distinguish between the two curves we use a value based on the mean and variance of the edges under consideration.

Curve fitting In this method we try to fit the edge weights to the weighted average of two normal distributions. We use five parameters – mean and standard deviations of the intra-cluster and inter-cluster edges, and the proportion of the edges that are intra-cluster edges. We estimate the distributions using the Levenberg-Marquardt method, which is described in Numerical Recipes [9]. We first give an initial guess of the parameters to the Levenberg-Marquardt procedure, and then it iterates, trying to minimize the χ-squared value of the fit between the edge weights being clustered and the cumulative distribution function defined by the five parameters described above.

Probabilistic In the probabilistic thresholding method, each edge is assigned an estimate of the probability that it is an inter-cluster edge, and the edges are then thresholded, randomly and independently, according to these probability estimates. Specifically, simple estimates for the means and standard deviations of the inter- and intra-cluster edge weight distributions are computed, and these parameters are used to determine the probability that any given edge is inter-cluster, under the assumption that the distributions are normal; edges are then thresholded, randomly and independently, with these probabilities. Details are provided in the full paper.

Step 2.5: Computing $C^{(t)}$ In step 3 we determine a set of clusters by computing the connected components of the graph generated at step 2. (See [3] for an introduction to graph algorithms.) When we threshold by vertex the resulting graph $M^{(t)}$ is not necessarily symmetric since edge ij can be thresholded while ji is not. In this case we have the following three choices:

1. Do not make $M^{(t)}$ symmetric after the thresholding step, and compute the connected components.
2. Do not make $M^{(t)}$ symmetric after the thresholding step, and compute the strongly connected components.
3. Make $M^{(t)}$ symmetric by thresholding an edge (i, j) if (j, i) was thresholded, and compute the connected components.

We will use the third option, and justify this choice as follows. It is not hard to argue that the first option is likely to perform badly, since a single inter-cluster edge will result in two clusters collapsing into one. For the other two cases we will examine the cases in which selecting one option over the other is disadvantageous, either by collapsing clusters which should be disjoint or by breaking a cluster into separate subclusters.

The drawback with the second method is that two clusters A and B can be collapsed as long as there is one edge from A to B and another one from B to A. With the third method A and B will be joined only when both ab and ba ($a \in A$ and $b \in B$) are present in $M^{(t)}$, which is less likely to happen if A and B are indeed disjoint clusters.

The third method can be bad if cluster A is subdivided into A_1, A_2 and A_3 such that there are only edges directed from A_1 to A_2, from A_2 to A_3 and from A_3 to A_1. But if this is the case it could indicate that there is some extra structure in A, and that maybe it should be seen as three separate clusters. For these reasons we have selected to make $M^{(t)}$ symmetric after the thresholding step.

Step 3: Preparing the next iteration After we have thresholded edges in $\bar{E}$, we have to set the weights of the remaining edges in order to compute clusters and start the next iteration. First for every edge ij such that ji is in $\bar{E}$, we also set $M_{ij}^{(t)}$ to 0. This will make $M^{(t)}$ symmetric. Then for every other $M_{ij}^{(t)}$ we have considered 4 different possibilities. Either $M_{ij}^{(t)}$ remains unchanged or we set $M_{ij}^{(t)}$ $= M_{ij}^{(0)}$. In addition, we can decide that $M_{ij}^{(t)}$ is always set to 0 if ij or ji had been threshold in a previous iteration.

3 Experimental results

In this section we describe preliminary results for different versions of our algorithm. We had three objectives with these experiments. First we wanted to see how the process of reweighting and thresholding would separate the intra-cluster and inter-cluster edges, leading to a useful algorithm. Second we wanted to know which of the several variations described in section 2 would give the best results. Finally we wanted to see how each variable (number of clusters, size of clusters, original means and variances) would influence the performance of the algorithm.

In the remainder of this section we will describe the data used and then analyze the results of the experimentations. In order to evaluate the results we need a quantitative measure to compare the clusters that we compute with what we think the clusters actually are. We will use an adaptation of the precision-recall method, which has been used for document clustering (see, for instance, [1])

3.1 Precision-recall computation

The objective of the precision recall computation is to give a quantitative evaluation of the output of a clustering algorithm. For this paper the output is compared with what we expect the clusters to be based on the parameters used to generate the input graph.

For a vertex v let $C(v)$ be the cluster containing v that was computed by an algorithm, and $\tilde{C}(v)$ what we expect the cluster containing v to be, either from an experts opinion in the case of real data, or from the model in the case of random data. Informally the precision of v says which proportion of the vertices in $C(v)$ should really belong in a cluster with v. Conversely the recall value of v indicates the proportion of the vertices that should be in a cluster with v that have been assigned to $C(v)$ by the algorithm.

We will define the precision and recall values for a clustering to be the expected precision and recall values of a vertex randomly picked from the graph. More formally, the precision value for v is defined as $P(v) = (|C(v) \cap \tilde{C}(v)|)/|C(v)|$. The recall value for v is defined as $R(v) = (|C(v) \cap \tilde{C}(v)|)/|\tilde{C}(v)|$. For a complete set S, we define the precision of the clustering Ω of the algorithm on the set S as $P(\Omega) = (\sum_{v \in S} P(v))/|S|$, and the recall as $R(\Omega) = (\sum_{v \in S} R(v))/|S|$.

The precision and recall values are two complementary measures, each of which can be high on clusterings that are obviously wrong. If all the clusters are singleton then the precision is 1. If we have only one cluster containing all the vertices then the recall is 1. To handle this problem we define an error measure $E(\Omega) = ((1 - P(\Omega))^2 + (1 - R(\Omega))^2)/2$, which is the mean squared error. The error is 0 when the clustering is perfect.

3.2 Methodology

We implemented the program as described in Figure 1.

There are 24 different variations of the algorithm, and we tested them all. We first fixed the distribution parameters, and then for each set of parameters we generated a random graph based on these parameters, and ran all 24 variations of the algorithms. We will describe each variation with a 4 letter code, such as VCYO, where the first character is is V if the thresholding is done vertex by vertex, and G if it is done globally; the second cahracter is H for the histogram method, P for probabilistic, and C for curve fitting; the third character is Y or N, for whether the thresholded edges can be set back to their original value or not; and the last character is O if the non-thresholded edges are set to their original value, or L if they are set to the value computed at the last iteration.

We measured the performance of a variation of the agorithm on a data set by taking the iteration t for which C^t minimizes the error value as described in section 3.1.

The data we used came from three different types of distribution: the normal, uniform, and bimodal (every edge has weight $\mu - \sigma$ or $\mu + \sigma$) distribution. In every case both the intra-cluster and inter-cluster edges had the same type of distribution, although with different parameters. We generated random graphs where every edge is assigned a weight with a value sampled from the relevant distribution.

3.3 Summary of results

In the results that we present we generated at least 5 data sets for each setting of parameters, and averaged the error measure. The program executes 20 iterations. Limiting the number of iterations to 20 does not affect the analysis, at least on the data that we looked at so far. Except in very few cases the error value at the end of the iterations first decreases monotonically, then increases monotonically or reaches a steady state. The optimal value was reached before the 10th iteration in all but a few cases.

Comparison of methods We have executed the 24 different variations of the algorithm in order to know if any would perform better that the others. We then collected all the data and averaged the error values. The results are presented in table 1. For each entry in the first 3 columns we have executed the algorithm described in the left cell on more than 150 random data sets with different parameters using the distribution indicated by the column label. In the last column we combine the results from all the distributions. By looking at the table it appears that the best methods are VCNL and VHYO, and that the histogram method is in general more robust.

Other interesting observations that we draw from the results are:

1. Thresholding vertex by vertex yields better results than thresholding globally. A possible explanation for this is that a different thresholding point is selected for the edges adjacent to each vertex. Therefore selecting a bad thresholding point will impact only a few edges at time, while a bad global thresholding point could result in removing either far too many or too few edges.
2. The histogram method works best when we always use the original edge weights to start an iteration and when we allow an edge that has been thresholded to be assigned a value later. But the other two methods performs best when we use the last weights computed and set to 0 the weight of any edge that was thresholded in a prior iteration. We have no explanation for this.

To gain a better intuition of what the error value means, consider the case where we have 6 clusters of size 50, which is a typical data set that we experimented with. When the precision and recall values are 1 then the error is 0. When the error is near 0.083, the average for our best algorithms, the precision-recall values from our experiments that produced an error near this value were (0.9729, 0.5956), (0.6076 0.8912), (0.5952 0.9803). If we consider an error of 0.23, which is in the middle of our table, then typical precision-recall values that produced this error were (6344, 0.4285), (0.5521, 0.4961) and (0.4095 0.6808).

If we have only one cluster then the recall is 1 while the precision is 0.1667, giving an error value of 0.3472. Conversely if every cluster is a singleton then the precision is 1 while the recall is 0.022, resulting in an error value of 0.4782. Therefore there is some significant difference in performance depending on which version of the algorithm we use. The best ones perform very well while the worst ones produce an error value not very far from what one would obtain by always generating one cluster containing all the vertices. This difference in performance can not be explained by the implementation choice of one parameter alone, as attested by the fact that reusing the original edge weight at each iteration is helpful with the histogram method but not with the curve-fitting method.

Normal		Bimodal		Uniform		Combined	
Method	Avg error	Method	Avg error	Method	Avg error	Method	Avg error
VCNL	0.084	VHYO	0.069	VHYO	0.089	VCNL	0.083
VHYO	0.090	VHYL	0.069	VCNL	0.091	VHYO	0.083
VHYL	0.095	GHYO	0.069	VHYL	0.107	VHYL	0.091
GHYO	0.101	GHYL	0.072	GHYO	0.111	GHYO	0.094
VPNL	0.106	VCNL	0.072	VPNL	0.116	GHYL	0.101
GHYL	0.110	VPNL	0.100	GHYL	0.119	VPNL	0.108
VHNL	0.119	VHNL	0.103	VHNL	0.123	VHNL	0.115
VHNO	0.137	VHNO	0.131	VHNO	0.175	VHNO	0.147
GHNO	0.231	GHNL	0.142	GHNO	0.214	GHNO	0.202
VCNO	0.232	GHNO	0.158	GHNL	0.225	GHNL	0.208
VPYL	0.235	VCYO	0.196	VPYO	0.230	GPYO	0.228
GPYO	0.238	GPYO	0.205	VCYL	0.236	VCYL	0.230
VCYL	0.239	VCYL	0.214	VPYL	0.239	VPYL	0.231
GPYL	0.241	GPYL	0.215	GPYO	0.240	VCYO	0.232
GCYL	0.245	VPYL	0.219	GPYL	0.243	GPYL	0.233
VPYO	0.247	GPNL	0.222	GPNL	0.247	VPYO	0.235
GCYO	0.247	GCNL	0.225	GCYO	0.247	GPNL	0.240
GPNL	0.249	GCYO	0.226	VCYO	0.248	GCYO	0.240
GHNL	0.250	GCYL	0.229	GCYL	0.250	GCYL	0.241
VCYO	0.253	VPYO	0.229	GCNL	0.256	GCNL	0.251
GCNL	0.272	VCNO	0.268	VCNO	0.276	VCNO	0.259
GCNO	0.298	GPNO	0.318	GPNO	0.298	GPNO	0.305
GPNO	0.298	GCNO	0.319	GCNO	0.300	GCNO	0.305
VPNO	0.307	VPNO	0.327	VPNO	0.314	VPNO	0.316

Table 1. Comparison of the performance of the 24 different variations of the algorithm. Each entry is obtained by computing the average of at least 150 error values.

Influence of input parameters We have also looked at how the performance of the algorithm changed as we varied different parameters. The most important parameters that can influence the performance are the type of distribution, the size of the clusters n, the number of clusters k, and the relative values of the intra- and inter-cluster edge weight distributions $D_1(\mu_1, \sigma_1)$ and $D_2(\mu_2, \sigma_2)$. We now describe what we observed from our experiments.

The size and number of clusters have a significant impact on the performance. The performance of the algorithm will improve as the size of the clusters increases and as the number of clusters decreases. This is to be expected because in the computation of the cosine, the influence of the intra-cluster edges will diminish as the number of clusters increases. In section 4 we will show that when all the other parameters are fixed we can always find all the clusters in one iteration, for large enough n.

The original distributions also play a major role. The most important determinant is the ratio $\frac{\mu_1 - \mu_2}{\sigma}$ (for $\sigma = \sigma_1$ or σ_2). The performance improves as this ratio increases, since a greater value of this ratio implies fewer inter-cluster edges with a high value and/or fewer intra-cluster edges with a low value.

We have included a few plots that illustrate our results. All those plots were obtained by executing our algorithm on data that is uniformly distributed. We only

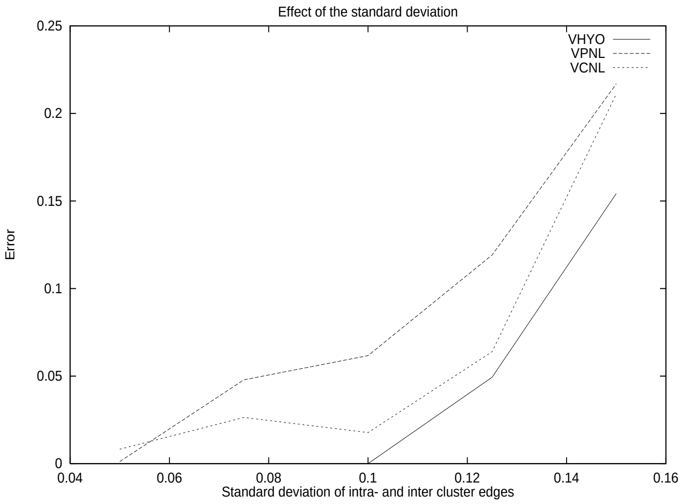

Fig. 2. Change in error value as we increase both standard deviations which are equal. There are 6 clusters, each of size 50. The intra-cluster mean is 0.7, the inter-cluster mean is 0.6.

included the data for the best performing variation that use each of the histogram, probabilistic and curve-fitting thresholding method according to table 1. In each case the error value is averaged over at least 5 different data sets. Figure 2 shows how the performance decreases as the standard deviation increases. We obtain a similar graph if we move the means closer while keeping the standard deviations fixed. Figure 3 shows the effect of increasing the number of clusters. Figure 4 illustrates how the performance increades as we increase the cluster size. Notice that in the last case each cluster has a different intra-cluster distribution.

4 Explanation of results

A first insight into why the method introduced in this paper will work comes from the following. To simplify, consider the case of k clusters of size n, with two arbitrary distributions $D_1(\mu_1, \sigma_1)$ for the intra-cluster edges and $D_2(\mu_2, \sigma_2)$ for the inter-cluster edges. We are interested in the distributions $\tilde{D}_1(\tilde{\mu}_1, \tilde{\sigma}_1)$ and $\tilde{D}_2(\tilde{\mu}_2, \tilde{\sigma}_2)$ of the intra-cluster and inter-cluster distributions after executing step 1 of the algorithm described in Figure 1. A key observation is the following:

Observation 1. *In most cases, after step 1 of the algorithm, both $\tilde{D}_1$ and $\tilde{D}_2$ are normally distributed, or at least very close to be normally distributed.*

We will attempt to explain this phenomenon. First consider a simpler version of step 1 where we compute only the dot products of every pair of vectors without normalization. Let $D'_i(\mu'_i, \sigma'_i)$ denote the intra and inter-cluster edge distributions resulting from computing the dot products. Using the Central Limit theorem and basic probability theory (see for instance [5]) we observe that D'_1 forms a normal

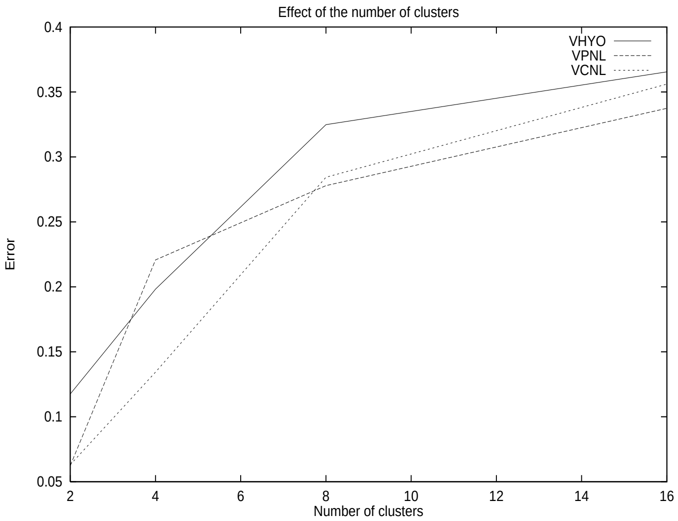

Fig. 3. Change in error value as we vary the number of clusters. The clusters contain 30 vertices. The intra-cluster mean is 0.7, the inter-cluster mean 0.6, and both standard deviations are 0.15.

distribution with mean $n\mu_1^2+(k-1)n\mu_2^2$ and variance $n(\sigma_1^4+2\mu_1^2\sigma_1^2)+(k-1)n(\sigma_2^4+2\mu_2\sigma_2^2)$ while D_2' forms another normal distribution with mean $2n\mu_1\mu_2+(k-2)n\mu_2^2$ and variance $2n(\sigma_1^2\sigma_2^2+\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)+(k-2)n(\sigma_2^4+2\mu_2^2\sigma_2^2)$. From this we obtain

$$\frac{\mu_1'-\mu_2'}{\sigma_2'} = \frac{\sqrt{n}(\mu_1-\mu_2)^2}{\sqrt{2(\sigma_1^2\sigma_2^2+\mu_1^2\sigma_2^2+\mu_2^2\sigma_1^2)+(k-2)(\sigma_2^4+2\mu_2^2\sigma_2^2)}} \tag{1}$$

The value in equation 1 indicates how far away μ_1' is from μ_2' as a multiple of σ_2. The main objective of step 1 of the algorithm is to maximize this value. When using only the dot products n has to be huge relatively to k in order for the means to be further apart after the transformation than before.

However when we normalize the vectors of the input matrix before we compute the dot product the ratio in equation 1 can be relatively big with much smaller values of n. To illustrate consider the case of $k=6, n=50, \mu_1=0.7, \mu_2=0.6, \sigma_1=\sigma_2=0.1$. After one iteration using the dot products we obtain $(\mu_1'-\mu_2')/\sigma_2' \approx 0.5$, in which case we are worse than we were before the transformation. However if we normalize we get $(\tilde{\mu}_1-\tilde{\mu}_2)/\tilde{\sigma}_2 \approx 2.5$. We can notice from equation 1 that for n large enough, when all the other parameters are fixed, we can find all the clusters with high probability after one iteration since the ratio $(\mu_1'-\mu_2')/\sigma_2'$ grows linearly with $\sqrt{n}$.

We have just shown formally that D_1' and D_2' form normal distributions. We would like to be able to do the same for $\tilde{D}_1$ and $\tilde{D}_2$. While it is not the case that $\tilde{D}_1$ and $\tilde{D}_2$ always form a normal distribution, we have significant supporting evidence to show that they are, in many cases, nearly normal. First, we have performed extensive simulations to show that the resulting distributions $\tilde{D}_1$ and $\tilde{D}_2$ look normal. Second, we compute $\tilde{D}_1$ by dividing each element of $D_1' = M_iM_j$ by a

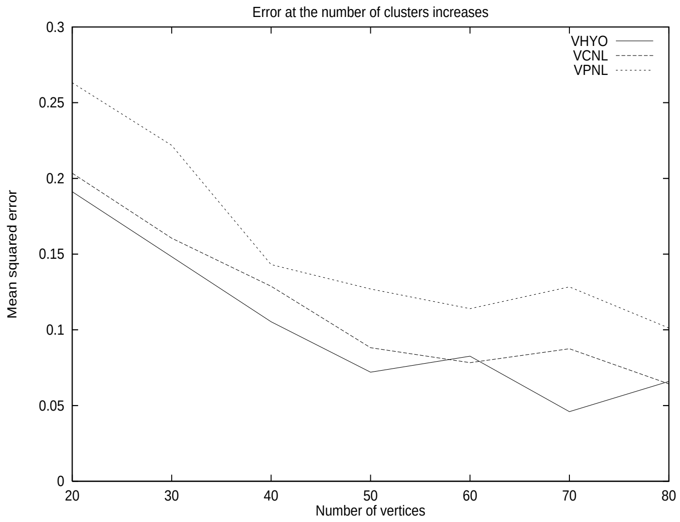

Fig. 4. Change in error value as we vary the size of the clusters where we the intra-edge distributions vary among the clusters. The inter-cluster distribution is U(0.5, 0.2). There are 6 clusters with distributions: U(0.6, 0.15), U(0.6, 0.15), U(0.7, 0.2), U(0.7, 0.25), U(0.8, 0.3), U(0.8, 0.35)

normalization factor which is $||M_i|| \cdot ||M_j||$. Therefore the normalization factor used is not the same for all the elements of D_1'. However the normalization factors form a distribution which is also close to a normal, has the same order of magnitude as the mean of μ_1', but which has a much smaller variance. Therefore $\tilde{D}_1$ is not very different from what we would obtain if we were to divide all the elements in D_1' by a constant factor.

5 Conclusion and future work

We have presented a new method to cluster a data set based on the idea of separating the intra-cluster edges from the inter-cluster edges by analyzing the edge weight distribution. We have proposed and implemented several procedures based on this idea. We have also presented experimental results on non-trivial random data where the algorithm computes a perfect or near-perfect clustering in many cases. This method can work on data that is very noisy, and the algorithm has no prior knowledge of the expected number of clusters or of what the inter-cluster and intra-cluster distributions of the input graph are.

The next step is for the algorithm to be able to output a final set of clusters, as opposed to generate one at the end of each iteration. We have implemented an algorithm to perform this task and the first experimental results are encouraging. In addition to making the algorithm useful to cluster real data, this new algorithm should also improve on the results presented in section 3 by combining the clusters generated through all the iterations.

On the theoretical side we would like to refine the analysis from section 4, and be able to explain why some variations of the algorithm perfrm better. We would also like to determine the limits of this algorithm. For instance how large can the standard deviations be, with all the other parameter constant, before we lose any hope of finding cluster with our algorithm, or in fact with any algorithm.

6 Acknowledgement

The authors wish to thank Bob Gross, Pablo Tamayo and Michael Angelo for helpful discussions.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.
2. A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4), 1999.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.
4. B. S. Everitt. *Cluster Analysis.* Oxford University Press, 1993.
5. P. G. Hoel, S. C. Port, and C. J. Stone. *Introduction to Probability Theory.* Houghton Mifflin, 1971.
6. L. Kaufman and P. J. Rousseeuw. *Finding groups in data.* John Wiley & Sons, Inc, 1990.
7. T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9), September 1990.
8. B. G. Mirkin. *Mathematical classfication and clustering.* Kluwer Academic Publishers, 1996.
9. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes.* Cambridge University Press, 1986.
10. G. Salton, A. Wong, and C. S. Yang. A vector space model for information retrieval. *Communications of the ACM*, 18(11):613–620, 1975.