Project Report on

*Effective and Efficient computation of ‘Cluster Similarity’*

Mayuresh P. Kunjir    Mangesh D. Pujari
{mayuresh,mangesh}@csa.iisc.ernet.in
M.E. (CSA)
Indian Institute of Science, Bangalore

Project Guide
Dr. Shirish Shevade

April 15, 2009
Abstract

In many algorithms in the field of data mining to perform clustering of given data, notion of ‘cluster similarity’ is used to a large extent. Given a set of clusters, we may want to merge them to form a smaller number of clusters; particular applications being Hierarchical clustering algorithms and ideas like SPARCL which is the motivation behind this project. We study some methods for computing the cluster similarity that are based on average linkage, K-nearest neighbors and SPARCL. We suggest some simple but effective variations to methods based on K-nearest neighbors and SPARCL which perform better than basic methods. To measure accuracy of similarity given by these methods, we propose two simple metrics which capture the closeness of similarities given by different methods and can be a good estimate of correctness. Our experimental analysis shows that outputs given by KNN based similarity and its variation KNNCEN and variation of SPARCL give accurate results and our proposed idea KNNCEN does the computations fastest among all.

1 Introduction

Given the clustering of a data set, cluster combining is the activity of merging the clusters which are similar to each other. The key task in cluster combining is to compute the similarities between the clusters. The clustering algorithms like Agglomerative Hierarchical Clustering use the notion of cluster similarity to combine the clusters. In this project, we study different algorithms to do this task of computing similarity among clusters effectively and efficiently. The basic algorithms we considered are:

1. Average Linkage based similarity (A-Link)
2. K-Nearest Neighbors based similarity (KNN)
3. SPARCL approach

We also give some variations of these basic methods namely KNN with centroid consideration (KNNCEN) and SPARCL with fixed number of bins (SPARCL-n). All suggested methods take a clustering of data with $K$ clusters as input and produce $K \times K$ matrix denoting the pair wise similarity among the clusters. We implemented all the basic methods and variations and performed some experiments on time efficiency and accuracy of these methods. Accuracy of any particular similarity matrix can’t be measured using any formula or set rule. So to measure the accuracy of similarity matrix computed, we suggest two metrics viz. Variance and Top – Error which measure relative closeness between the output produced by different methods and thus give a fair idea about which methods will produce good clusters and which will not.

Our experimental results show that a variation of SPARCL algorithm which we call SPARCL-n and a new idea proposed in section 2.6 called KNNCEN which is a modification of KNN based similarity outperform all other methods in terms of both efficiency and accuracy. These two methods along with KNN based similarity give fairly accurate results, while KNNCEN runs the fastest among all.
1.1 Motivation

Our motivation behind this project is the SPARCL (Shape Based Clustering) clustering method proposed by Zaki et al. [1] which gives the efficient and effective method for producing arbitrary shape clustering of high dimensional data. Many different methods like spectral, partitional and mixture model suffer when dataset is large. SPARCL exploits the linear run time of K-means clustering algorithm while avoiding its drawback. It runs the K-means algorithm on given data first and finds the similarity matrix for the clustering produced by K-means. Then it gives this similarity matrix of clusters as input to some hierarchical or spectral clustering algorithm to produce arbitrary shape clusters by merging the clusters produced by K-means.

Also in Agglomerative or Hierarchical clustering algorithms like [3, 4], we produce many clusters of smaller sizes and then iteratively merge them to produce bigger clusters. These algorithms need some measure of cluster similarity to combine the clusters. This is another motivation behind this project.

1.2 Related Work

CURE [2] is a hierarchical agglomerative clustering algorithm that handles shape-based clusters. It follows the nearest neighbor distance to measure the similarity between two clusters, but reduces the computational cost significantly by taking a set of representative points from each cluster and engaging only these points in similarity computations. CURE is still expensive with its quadratic complexity, and more importantly, the quality of clustering depends enormously on the sampling quality.

CHAMELEON [3] also formulates the shape-based clustering as a graph partitioning algorithm. A m nearest neighbor graph is generated for the input dataset. This graph is partitioned into a predefined number of sub-graphs (also referred as subclusters). The partitioned sub-graphs are then merged to obtain the desired number of final k clusters. CHAMELEON introduces two measures: relative interconnectivity and relative closeness that determine if a pair of clusters can be merged. Sub-clusters having high relative closeness and relative interconnectivity are merged. But this algorithm becomes very expensive when the dataset size becomes considerably large, since computing the nearest neighbor graph can take $O(n^2)$ time as the dimensions increase.

SPARCL [1] gives a completely different method to compute similarity between two clusters by projecting the points of the clusters on line joining centers of the clusters. The similarity is computed taking into consideration densities, distance and width of separating hyper-plane of two clusters. Though this approach produces good quality of clusters, it doesn’t scale too well with dimensionality of data as projection operation takes lot of time if number of dimensions is high.
1.3 Organization

The rest of project report is organized as follows; Section 2 introduces the idea of all algorithms mentioned above in detail. Two variations of the basic methods are also proposed in this section. In section 3, two metrics are suggested to measure relative closeness or accuracy of our methods. The experimental evaluation of all the methods is done in section 4, where we first show correctness of methods on a toy dataset and then accuracy is measured using our metrics for larger datasets. Also we experiment on time efficiency of the methods by checking the effect of changing number of dimensions as well as number of examples. Finally in section 5, we give final conclusions and future work.

2 Approaches to compute Cluster Similarity

We study some approaches to compute similarity among clusters in this section. First, three basic methods are given and then three variations are suggested. In all, we study following methods here:

1. A-Link (Average Linkage)
2. KNN (K-Nearest Neighbors based similarity)
3. SPARCL
4. SPARCL-n (SPARCL with fixed bins)
5. KNNRP (Approximate K-Nearest Neighbors by random projection)
6. KNNCEN (Approximate K-Nearest Neighbors by centroid considerations)

We will see algorithms and description for each of these approaches in this section.

2.1 A-Link (Average Linkage)

This is the natural idea that comes to mind first while deciding which clusters to merge. Average linkage is calculated by taking into account distance between all pairs of clusters. The distance is just the Euclidean distance between centroids of the pair under consideration. The shorter is the distance, higher the similarity between clusters should be and vice versa. So we just use the inverse proportion between distance and corresponding similarity to fill similarity matrix. The algorithm to do it is given below.

1. Get all data points along with cluster labels.
2. Find centroids of all clusters.
3. For each pair of clusters \(X\) and \(Y\)
   a. Find Euclidean distance(due) between centroid(\(X\)) and centroid(\(Y\))
   b. similarity(\(X, Y\)) = \(\frac{1}{\text{dist}}\)
4. End for
5. Normalize similarity matrix.
2.2 KNN (K-Nearest Neighbors based similarity)

This is a completely different approach than average linkage. Here we consider only nearest data points to decide the closeness among clusters. So in a way this method is closer to single linkage. For every cluster, we consider nearest \( K \) data points from every data point of the cluster and the distances between points and their cluster labels decide the closeness (i.e. similarity score). As can be seen from the algorithm given below, this method does a lot of computations (proportional to \( O(n^2) \)).

1. Get all data points along with cluster labels.
2. Calculate \( K \) nearest neighbors of all data points.
3. For each cluster \( X \)
   
   For each point \( P_1 \) in the cluster,
   
   For all \( k \)-nearest neighbors of \( P_1, P_2 \)
   
   1. Let \( Y \) is the cluster label of the neighbor \( P_2 \).
   2. \( \text{similarity}(X, Y) + = \text{distance}(P_1, P_2) \)
   
   End for
   
   End for

4. End for

5. Normalize similarity matrix.

2.3 SPARCL(Shape Based Clustering)

SPARCL is a simple and scalable algorithm for finding clusters with arbitrary shapes and sizes, and it has linear space and time complexity. SPARCL consists of two stages: The first stage runs a carefully initialized version of the K-means algorithm to generate many small seed clusters. The second stage iteratively merges the generated clusters to obtain the final shape-based clusters.

Figure 1: Projection of points onto line between centers
To compute similarity between clusters produced by K-means algorithm, SPARCL gives a similarity score to a pair of clusters by computing size similarity and distance similarity between them. It gives a high similarity if following conditions are satisfied:

1. The clusters are close to each other in the Euclidean space.
2. The densities of the two clusters are comparable, which implies that one cluster is an extension of the other.
3. The face (hyper-plane) at which the clusters meet is wide.

For computing the similarity, points belonging to the two clusters are projected on the vector connecting the two centers as shown in Figure 1. The algorithm to compute similarity between two given clusters $X$ and $Y$ is given below:

1. For cluster $X$
   a. For every point in cluster $X$
      i. Project it on line joining $\text{centroid}(X)$ and $\text{centroid}(Y)$
   b. End for
   c. Calculate mean ($m_X$) and standard deviation ($\sigma_X$) of projected distances.
   d. Form bins of width $= \sigma_X/2$, number of bins formed $= |B_X|$
   e. For every projected point
      i. Assign it to a bin $X_i$ according to projected distance
      ii. Increment number of points in bin $X_i$, $N(X_i)$ by 1.
   f. End for
   g. $\text{max}_\text{bin} =$ bin having maximum number of projected points
   h. For every bin $X_i$
      i. $\text{dist}(X_i) =$ average horizontal distance of all points in bin $X_i$
      ii. $\text{size}\_\text{ratio}(X_i) = \frac{N(X_i)}{N(X_{\text{max}_\text{bin}})}$
2. End for
3. For cluster $Y$
   Find $\text{size}\_\text{ratio}(Y_i)$ for every bin $Y_i$ where $i : 1$ to $|B_Y|$ similarly.
4. End for
5. Let $t = \min(|B_X|, |B_Y|)$
6. For bins $i = 1$ to $t$
   a. Size similarity is given by
      $\text{size}\_\text{sim}(B_{X_i}, B_{Y_i}) = \text{size}\_\text{ratio}(B_{X_i}) \times \text{size}\_\text{ratio}(B_{Y_i})$
b. Distance similarity is given by,

\[ \text{distsim}(B_X, B_Y) = \frac{2 \times \text{dist}(B_{X_i}, B_{Y_i})}{\sigma_{X} + \sigma_{Y}} \]

7. End for

8. Overall similarity between clusters \(X, Y\) is given by

\[ S(X, Y) = \sum_{i=1}^{t} \text{size\_sim}(B_{X_i}, B_{Y_i}) \ast \exp^{-\text{distsim}(B_{X_i}, B_{Y_i})} \]

This approach given in SPARCL paper satisfies above mentioned three conditions by giving more weights to the points nearer to cluster boundary as can be seen from distance similarity formula. Also it considers density of clusters by calculating size similarity which is high if densities of two clusters are similar. So this method should in theory produce good quality of clusters. Computationally though it doesn’t scale too well with increasing number of dimensions as projecting the points on the line becomes computationally very expensive in that case.

2.4 SPARCL-n

We observed through our experiments that in the SPARCL approach to compute similarity, the width of the buckets is too high. They fix it to be half of the standard deviation which is observed to be very high for the random datasets we considered. This results in very low number of bins (in fact, for all the experiments that we did with SPARCL; never more than two bins were used). This kills the whole purpose of SPARCL to weigh similarity score based on the closeness among the points. So we suggest a simple variation which is to keep number of bins fixed at a sufficiently large value so that we can have more weights given to the points near the boundary of two clusters and lesser weights for the points in the interior of the clusters. This value needs to be decided based on experiments.

2.5 KNNRP (KNN with random projection)

The general KNN algorithm is computationally very inefficient as can be seen from the algorithm given above. So some method is needed to approximate KNN. The first idea that comes to mind is random projection which is to project all the points to a smaller dimensional space to remove the curse of dimensionality and then calculate \(K\)-nearest neighbors of all points by considering this smaller dimensional space. In particular, the data matrix of dimensions \((N \times d)\) is multiplied by a random projection matrix of dimensions \((d \times d')\) where \(d' \ll d\), to get a matrix of dimensions \((N \times d')\) which will be used further.

2.6 KNNCEN (KNN with centroid consideration)

Although approximation by random projection is a simple idea, it is a huge compromise with accuracy and also we are still computing nearest neighbors of all data points. So random projection may not result in much time improvements compared to nearest neighbors. We here suggest a smart idea to achieve both good accuracy and efficient execution. Instead of calculating neighboring clusters by considering all the data points of a cluster, we consider only
centroid of the clusters which is good representative of all data points, unlike in [2], where a random sample of points is used as representatives. So for a given cluster, we compute \( k \)-nearest points from centroid of the cluster instead of all points of the cluster thereby reducing the time complexity to \( O(KNd) \) where \( K \) denotes number of clusters given as input to KNNCEN algorithm. We compute similarity score based on distances and cluster labels of these neighbors. This method drastically reduces number of calculations and also results in better output compared to random projection. This fact is verified by experimental results.

1. Get all data points along with cluster labels.
2. Find centroids of all the clusters.
3. For every cluster \( X \)
   a. For each data point not in cluster \( X 'P1' \)
      Find Euclidean distance from centroid(\( X \)) to \( P1 \).
   b. Choose \( k \)-nearest centroids among these.
   c. For all \( k \)-nearest neighbors \( 'P2' \)
      1. Let \( Y \) is the cluster label of the neighbor \( P2 \).
      2. \( \text{similarity}(X, Y) = \text{distance(centroid}(X), P2) \)
   d. End for
4. End for
5. Normalize similarity matrix.

3 Metrics to measure accuracy

As we explained in the last section, there is no formula or rule to measure accuracy of our similarity matrix. We instead compare results produced by different methods. If the outputs given by two methods differ by a huge amount, there is a great chance of one being correct result and other being incorrect. Now if output of a method gives very different results than all other outputs, we can conclude that this output is probably wrong. We give two metrics to measure relative closeness between two similarity matrices below.

3.1 Variance

To compute the difference between given two similarity matrices, we suggest measuring variance of the difference matrix of the two matrices which is formed using:
\[
difference(i, j) = \text{similarity1}(i, j) - \text{similarity2}(i, j)
\]
for all entries \((i, j)\) in similarity matrices \( \text{similarity1} \) and \( \text{similarity2} \).
The difference matrix entries should be close to each other if the two similarity matrices under consideration are similar i.e. the similarity score assigned to every pair of clusters by both similarity matrices should be in the same relative order. So variance of the difference matrix should be close to zero in this case. If the variance is high, the two outputs differ in many entries of matrix. So variance of difference matrix gives a good indication of the dissimilarity between two matrices.

3.2 Top-Error

Variance measure doesn’t always give good estimate of accuracy between different similarity matrices. As is the case, we don’t have any ‘accurate similarity matrix’ to which we can compare our matrices to conclude anything about their accuracy. But the one thing which most of the outputs should agree upon is the similarity value assigned to the clusters which are really close. The clusters which are really close should be given high similarity score by every similarity matrix computing method. We exploit this fact to compute ‘Top-Error’. Some definitions are given below which make the things clearer.

**Definition 1: Critical entry**

We call an entry of similarity matrix critical if the similarity score for that particular entry is among the top \( p \) similarity scores of the matrix or in other words the critical entry corresponds to the two clusters that are close and will be considered for merging.

**Definition 2: Top-Error value**

Top-Error value of a pair of similarity matrices is the number of entries out of \( p \text{critical} \) entries of one matrix which are not marked as critical by other matrix.

So Top-Error value gives the number of entries out of all critical entries where two similarity matrices differ. Suppose we have two similarity matrices \( A \) and \( B \) which are given as below:

\[
A = (0, 0.8, 0.4), (0.9, 0, 0.5), (0.5, 0.6, 0) \\
B = (0, 0.9, 0.6), (0.9, 0, 0.3), (0.3, 0.4, 0)
\]

If we are looking at 3 critical entries, the Top-Error = 2 for this example as the critical entry (3,2) of matrix A is not critical in matrix B and critical entry (1,3) of matrix B is not critical in matrix A.

4 Experimental Study and Results

The two important features that any good cluster similarity computing method should have are:

1. Accuracy or Correctness
2. Time Efficiency

We first do experiments on accuracy and then on time efficiency for all the methods given above.
4.1 Accuracy measurements

There is no hard and fast rule to decide whether similarity matrix has produced correct results. Any similarity matrix may be better than other in some situations. In some cases, giving more weight to points near the boundary may result in better clustering while in other cases you may want to check average distance between clusters. Accordingly, different similarity computing method may work better than other. We first show clustering results using similarity matrices given by all suggested methods on a sample two-dimensional dataset and give results on all test datasets using our proposed metrics.

4.1.1 Clustering on a toy dataset

We first tested on a small dataset to compare similarity matrix resulting from each method. Specifically, we applied $k$ – means algorithm on a test data of 75 points which is in two-dimensions to produce 12 clusters. The output produced by $k$ – means is shown in figure 2. Now all methods are given this data along with cluster labels as input and similarity matrix thus produced is tested. We considered a test scenario where we want only 4 clusters as final output. So looking at similarity matrix, we tried to merge produced clusters using ‘greedy approach’ so as to produce 4 clusters in the output. We fixed a confidence level threshold to 20% of the maximum similarity score of the matrix. Whenever we greedily choose a pair of clusters to merge, we test whether similarity score is above the confidence value and only if it is, we proceed to merge them.

Using the similarity matrix produced using any method, we try to find best possible 4 clusters. The outputs are given in the Figures 3a-3f. Different colors are used for each of the 4 clusters thus formed. The clusters for which confidence value is below our set threshold level are not colored, i.e. the given cluster is not shown to be too closer to any other cluster to merge. We used 10 bins for

---

1We pick up the highest valued entry from the similarity matrix which is above some threshold value and merge the clusters corresponding to this entry
It can be observed that methods A-Link, KNN and KNNCEN produce good clustering. There are some differences in the clusters that are chosen to be merged by each method though due to different similarity measures. The methods KNNRP and SPARCL produce undesirable results as can be observed from the figure that some clusters that are chosen to be merged are really far away from each other. The output of A-Link is not desirable in many situations as it doesn’t consider distribution of data near the boundary of clusters which is a crucial parameter for any good similarity computing technique. Our variation SPARCL-10 doesn’t produce good result for this dataset since this dataset contains too few examples to use the weighted size similarity measure of SPARCL. For datasets containing large number of examples, SPARCL and SPARCL-10 are observed to do better which can be verified from the analysis in our next.
subsection. KNNRP, due to its randomness never produces good output and needs to be repeated many times to compute good similarity matrix.

4.1.2 Experiments on different datasets

We tested accuracy on different datasets having different number of dimensions and examples. All datasets are generated using MATLAB's Gaussian generator. The characteristics of the datasets are given in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>DS1</th>
<th>DS2</th>
<th>DS3</th>
<th>DS4</th>
<th>DS5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of examples (n)</td>
<td>600</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
<td>50000</td>
</tr>
<tr>
<td>Number of dimensions (d)</td>
<td>20</td>
<td>2</td>
<td>40</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Number of clusters (k)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 1: Datasets considered for experiments

The variance values are computed among all pairs of methods under all these datasets. In figure 4a and 4b, we show variance values under two datasets DS1 and DS4. We can observe from the figure that, KNNRP and SPARCL give high variance with all other methods while other methods among themselves do not give much variance. This means output similarity matrices produced by A-Link, KNN, KNNCEN and SPARCL-10 are similar and we can say they are close to accurate.

We now show results of Top-Error measure on different datasets. Figures 5a and 5b show the comparisons under datasets DS1 and DS5. We used 15 critical entries for DS1 and 30 critical entries for DS5. We have not shown results for KNNRP method as its variance comparisons show that its result is highly inaccurate and there is no point in testing it further. We got same observations under this measure also. Specifically, SPARCL is performing poorly here too.
and our variation SPARCL-10, KNNCEN, KNN and A-Link are all giving low errors. SPARCL-10 is the winner here with least error with most of the other methods.

4.2 Time Efficiency measurements

In this section, we show results of our experiments to check time efficiency of algorithms. We first show results on a toy dataset. Then we check the effect of increasing number of dimensions on different methods. Finally the effect of increasing number of examples is analyzed. For all the experiments, we used SPARCL-10 version as we found it to be giving fairly accurate results and for K-Nearest neighbors based techniques, 30 neighbors were used for dataset size of 5000 and 60 neighbors were used on dataset of 50000 examples.

4.2.1 Experiment on a toy dataset

We tested the time taken by each method by running algorithms on DS1 which has 600 examples with 10 dimensions and 12 clusters are produced by K−means. The graph in Figure 6 shows the time taken in seconds by each method to generate the similarity matrix. The SPARCL, SPARCL10 and KNNCEN methods are linear in size and dimension of data but since projection of data
points in 10 dimensions in both the SPARCL and SPARCL10 is time consuming task, these methods take much more time than KNNCEN. As the runtime of KNNRP and KNN is polynomial in size of data they both take more time, though KNNRP is faster than KNN due to its independence to dimension of data.

4.2.2 Effect of increasing number of dimensions

In figure 7, we show the effect of increasing number of dimensions of data. We checked for dimensions ranging from 2 to 100 and number of examples fixed at 5000. As shown in the graph the time taken by each method except KNNRP increases linearly with dimension, KNNRP is independent of dimension so its running time becomes linear with dimensions. Algorithms SPARCL,SPARCL10 and KNNCEN execute much faster than KNN and KNNRP although they are all linearly increasing with increase in dimensions.

![Graph showing effect of increasing number of dimensions](image)

4.2.3 Effect of increasing number of examples

We next show results by increasing number of examples of dataset in figure 8. Number of dimensions was fixed at 20 and number of examples were varied from 600 to 50000. Here methods show major variations. Algorithms KNN and KNNRP are very expensive as expected due to their quadratic time complexity with respect to number of examples. All other algorithms show linear behavior with time. KNNCEN runs the fastest and is observed to be 25% faster than SPARCL which is faster than SPARCL-10 due to uses smaller number of bins.
5 Conclusion and future work

In this project we studied three algorithms based on average linkage, $K$–nearest neighbors and SPARCL to find the similarity between the clusters. We also proposed to simple variations to two methods KNNRP, KNNCEN and SPARCL-n. KNN, KNNCEN and SPARCL-n all give fairly accurate results which we show through our two accuracy metrics. KNNRP and SPARCL give very inaccurate results. KNN executes in quadratic time with respect to size of input and it is linear in dimension. The runtime of KNNRP is independent of dimension of data but is still quadratic in size of data. Methods SPARCL, SPARCL-n and KNNCEN run in $O(KNd)$ time giving results fairly quickly where K is number of clusters that was given as input. Method KNNCEN gives the output fastest among all the methods. It is 25% faster than SPARCL on large datasets and on small datasets it is still faster. KNN, KNNRP take about same time as SPARCL on small datasets but are not scalable for large datasets due to their polynomial complexities.

The thing which is not analyzed in this project is how many neighbors are sufficient for $K$–nearest neighbors based techniques. In our experiments we used large enough neighbors to produce accurate results and were still able to show that our method KNNCEN works fastest. In the future, we would like to establish relation between number of neighbors required given size of data (i.e. number of examples and number of dimensions).

References


