Chapter 5

A Two-Phase Document Clustering Approach

In the previous chapter, a new document clustering algorithm using the concept of fuzzy sets is proposed, where each cluster is viewed as a fuzzy set over some finite universal set. The fuzzy representation of clusters provides a compact representation of all the data points in a particular cluster. However, the proposed algorithm had certain drawbacks. As each data point is read, it is compared with the existing clusters at that stage of the algorithm. The data point is merged with the cluster that satisfies a user defined threshold. That is, cluster representative is updated each time a data point is read. For this reason, the order of the input data may affect the clusters obtained. Due to this, some of the data points may be assigned to the wrong clusters. Another issue with representative based clustering approach is that the clustering quality deteriorates with increase in the size of the dataset as well as with the size of the clusters, as was observed in the experimental results. Also, the irregular shaped clusters cannot be detected. In this chapter, we propose a two-phase approach to the clustering problem of large dataset. In the first phase, which is the implementation of the proposed algorithm in the previous chapter, a single pass over the database is used to produce an in-memory summary of the data set. In the second phase, the in-memory summary of the data set obtained in the previous phase is
merged based on the concepts of neighbours. Most clustering algorithms for large datasets are two-phase.

5.1 Motivation

Until recently, most clustering algorithms were intended for datasets that fit in memory. But the amount of memory available is limited (typically, much smaller than the dataset size) whereas the dataset can be arbitrarily large, and the I/O cost involved in clustering the dataset should be minimized. There are several proposed clustering techniques for large datasets, for example BIRCH, CURE and CHAMELEON. Most of them are two-phase. That is, they first produce a compact in-memory representation of the dataset and then cluster that, rather than the original dataset.

BIRCH is a two-phase algorithm where it constructs an in-memory summary of the data distribution based on a new data structure called CF-tree. BIRCH deals with large datasets by first generating a more compact summary that retains as much distribution information as possible, and then clustering the data summary instead of the original dataset.

BIRCH summarizes a dataset into a set of sub-clusters to reduce the scale of the clustering problem. A Clustering Feature (CF) entry is a triple summarizing the information that we maintain about a sub-cluster of data points, denoted as:

$$CF = (N, \bar{LS}, SS)$$  \hspace{1cm} (5.1)

where $N$ is the number of data points in the cluster,

$\bar{LS}$ is the linear sum of the $N$ data points and
$SS$ is the square sum of $N$ the data points

Given the $CF$ entries of two disjoint sub-clusters, $CF_1 = (N_1, \overline{LS}_1, SS_1)$, and $CF_2 = (N_2, \overline{LS}_2, SS_2)$, according to the $CF$ Additivity Theorem the $CF$ entry of the sub-cluster that is formed by merging the two disjoint sub-clusters is:

$$CF_1 + CF_2 = (N_1 + N_2, \overline{LS}_1 + \overline{LS}_2, SS_1 + SS_2)$$

A CF-tree is a height-balanced tree with two parameters: branching factor ($B$ for nonleaf node and $L$ for leaf node) and threshold $T$. Each nonleaf node contains at most $B$ entries of the form $[CF_i, \text{child}_i]$, where $i = 1, 2, \ldots, B$, $\text{child}_i$ is a pointer to its $i$-th child node, and $CF$ is the $CF$ entry of the sub-cluster represented by this child. So a nonleaf node represents a sub-cluster made up of all the sub-clusters represented by its entries. A leaf node contains at most $L$ entries, and each entry is a CF. In addition, each leaf node has two pointers, 'prev' and 'next', which are used to chain all leaf nodes together for efficient scans. A leaf node also represents a sub-cluster made up of all the sub-clusters represented by its entries. But all entries in a leaf node must satisfy a threshold requirement, with respect to a threshold value $T$: the diameter (alternatively, the radius) of each leaf entry has to be less than $T$. The tree size is a function of $T$. The larger $T$ is, the smaller the tree is. We require a node to fit in a page of size $P$, where $P$ is a parameter of BIRCH. Once the dimension $d$ of the data space is given, the sizes of leaf and nonleaf entries are known, and then $B$ and $L$ are determined by $P$. So $P$ can be varied for performance tuning. Such a CF-tree will be built dynamically as new data objects are inserted. It is used to guide a new insertion into the correct sub-cluster for clustering purposes just as a B+-tree is.
used to guide a new insertion into the correct position for sorting purposes. However the CF-tree is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a sub-cluster (which absorbs as many data points as the specific threshold value allows).

*CURE* represents each cluster by a certain fixed number of points that are generated by selecting well scattered points from the cluster and then shrinking them toward the center of the cluster by a specified fraction. Having more than one representative point per cluster allows CURE to adjust well to the geometry of non-spherical shapes and the shrinking helps to dampen the effects of outliers. To handle large databases, CURE employs a combination of random sampling and partitioning. A random sample drawn from the data set is first partitioned and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters.

CURE maintains a set of representative points of each sub-cluster. The representative points are reasonably smaller in number to avoid the inefficiency of the all-points strategy. The representative points are well scattered within the sub-clusters so as to properly represent the whole sub-clusters, rather than the notational single point representative as in the case of centroid. In the event the sub-cluster being spherical in shape, the centroid is the best representative of the shape. But most real-life situations scattered representative objects of each sub clusters is the single most important aspect of CURE. The algorithm works as follows.

As with any agglomerative algorithm, it begins with every single data object as a cluster and the object itself is the sole representative of the corresponding cluster. At any given stage of the algorithm, we have a set of sub-clusters and associated with
each sub-cluster, we have a set of representative points. The distance between two sub-clusters is the smallest pair-wise distance their representative points. For every sub-clusters, $C$, its nearest sub-clusters $C_{nearest}$ is computed at this stage. Define a measure

$$D_{closest}(C) = \text{Distance}(C, C_{nearest})$$

The sub-clusters are arranged in the increasing order of $D_{closest}(C)$. Hierarchical clustering works by merging the closest pairs of sub-clusters. The sub-clusters $C$ corresponding to the smallest value of $D_{closest}(C)$. The sub-clusters $C$ corresponding to the smallest value of $D_{closest}(C)$ is the candidate sub-cluster to be merged with its nearest sub-cluster $C_{nearest}$ for the next stage. Once the clusters are merged, a new set of representative points are computed for the merged cluster. The merging process continues till the pre-specified number of clusters is obtained.

The problem now is to compute the set of representative points of a new sub-cluster. The set of representative points of individual sub-clusters may not represent the well-scattered set of points of the merged cluster. The method to compute the representative points of the newly-formed cluster is as follow.

It begins with the centroid of the new cluster and finds the farthest data object from the centroid. This is the first representative point. In the subsequent iterations, appoint in the sub-clusters is chosen, that is the farthest from the previously chosen representative points. The points are then shrunk towards the centroids by a fraction of $\alpha$. The idea of shrinking these points towards the centroid is to get over the problem of the outlier object as a scattered representative. CURE maintains a heap data
structure to determine the closest pair of sub-clusters at every stage. It also maintains a k-d tree for efficient implementation. While implementing the algorithm, one can think of many refinement and adjustments at each stage.

*CHAMELEON* is a novel hierarchical clustering algorithm called CHAMELEON that measures the similarity of two clusters based on a dynamic model. In the clustering process, two clusters are merged only if the inter-connectivity and closeness (proximity) between two clusters are high relative to the internal inter-connectivity of the clusters and closeness of items within the clusters. The merging process using the dynamic model presented in this paper facilitates discovery of natural and homogeneous clusters.

CHAMELEON uses an algorithm that consists of two distinct phases. The purpose of the first phase is to cluster the data items into a large number of sub-clusters that contain a sufficient number of items to allow dynamic modeling. The purpose of the second phase is to discover the genuine clusters in the data set by using the dynamic modeling framework to merge together these sub-clusters in a hierarchical fashion. In the remainder of this section, we present the algorithms used for these two phases of CHAMELEON.

**Finding Initial Sub-clusters** CHAMELEON finds the initial sub-clusters using a graph partitioning algorithm to partition the $k$-nearest neighbor graph of the data set into a large number of partitions such that the *edgecut*, i.e., the sum of the weight of the edges that straddle partitions, is minimized. Since each edge in the $k$-nearest neighbor graph represents the similarity among data points, a partitioning that minimizes the edge-cut effectively minimizes the relationship (affinity) among data points across the resulting partitions. The underlying assumption is that links within
clusters will be stronger and more plentiful than links across clusters. Hence, the data in each partition are highly related to other data items in the same partition.

CHAMELEON utilizes such multilevel graph partitioning algorithms to find the initial sub-clusters. In particular, it uses the graph partitioning algorithm that is part of the hMETIS library [22]. CHAMELEON obtains the initial set of sub-clusters as follows. It initially starts with all the points belonging to the same cluster. It then repeatedly selects the largest sub-cluster among the current set of sub-clusters and uses hMETIS to bisect. This process terminates when the larger sub-cluster contains fewer than a specified number of vertices.

**Merging Sub-Clusters using a Dynamic Framework** As soon as the fine-grain clustering solution produced by the partitioning-based algorithm of the first phase is found, CHAMELEON then switches to an agglomerative hierarchical clustering that combines together these small sub-clusters. The key step of agglomerative hierarchical algorithm is that of finding the pair of sub-clusters that are the most similar. CHAMELEON’s agglomerative hierarchical clustering algorithm utilizes the dynamic modeling framework to select the most similar pairs of clusters by looking both at their relative inter-connectivity and their relative closeness. Two different schemes have been implemented in CHAMELEON.

**The first scheme** merges only those pairs of clusters whose relative inter-connectivity and relative closeness are both above some user specified threshold.

**The second scheme** implemented in CHAMELEON uses a function to combine the relative inter-connectivity and relative closeness, and then selects to merge the pair of clusters that maximizes this function.
5.2 Proposed Two-Phase Clustering Approach

We next propose a two-phase clustering algorithm. The first phase is an implementation of the algorithm proposed in Chapter 4. It was observed that if the similarity threshold value is kept slightly higher then most of the clusters obtained were pure but the number of clusters was too high. The second phase is designed to merge these small clusters to obtain big clusters. The second phase is based on the concepts of neighbors and links which were introduced in ROCK [63]. A modified and quick version of the ROCK algorithm was proposed in QROCK [43]. The second phase proposed here is based on the works done in these two papers. For the sake of completeness a brief outline of the works done in these two papers is given below. After this, the second phase proposed by us has been described.

5.2.1 ROCK algorithm

The ROCK algorithm is an agglomerative hierarchical clustering algorithm based on the notions of neighbors and links. An object’s neighbors are those objects that are considerably similar to it. Given a threshold \( \theta \) (between 0 and 1) a pair of objects \( T_i \) and \( T_j \) is defined to be neighbors if \( \text{sim}(T_i, T_j) \geq \theta \). \( \text{Link}(T_i, T_j) \) between the objects is defined as the number of common neighbors between \( T_i \) and \( T_j \). If the \( \text{Link}(T_i, T_j) \) is large then it is more probable that \( T_i \) and \( T_j \) belong to the same cluster. ROCK clustering algorithm utilizes the information about links between points when making decisions about points to be merged into a single cluster. After an initial computation of the number of links between the data objects, the algorithm starts with each cluster being a single object and keeps merging clusters based on a goodness
measure for merging. The goodness measure $g(C_i, C_j)$ for merging two clusters $C_i, C_j$ is defined as

$$g(C_i, C_j) = \frac{\text{link}(C_i, C_j)}{\left(\frac{n_i + n_j}{\frac{1+2f(\theta)}{n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}}\right)}$$

(5.4)

where $\text{link}(C_i, C_j)$ is the sum of the cross links between the data objects in $C_i$ and $C_j$; $f(\theta) = (1+\theta)/(1-\theta)$;

$n_i$ and $n_j$ the sizes of the clusters $C_i$ and $C_j$.

The merging is continued till one of the following two criteria is met.

- a specified number of clusters is obtained or
- no links remain between the clusters.

ROCK requires that the dataset has to be memory resident. Therefore in case the database is large, ROCK draws random sample from the dataset. That is, instead of working on the whole dataset, ROCK clusters a sample randomly drawn from the dataset, and then partitions the entire dataset based on the clusters from the sample.

One issue with sampling is the choice of the sample size and sample number. Also the sample drawn may not be a good representation of the real dataset.

5.2.2 QROCK algorithm

The QROCK algorithm is essentially based on the same principle as that of ROCK. In ROCK, the agglomerative process of merging clusters terminates either when there is no pair of clusters with links between them or when the required number of clusters is obtained. The authors proved that if the merging of the clusters
is continued till no links remain between the clusters, then the final clusters obtained are nothing but the connected components of a graph with the input data objects as vertices and with two points connected by an edge if the pair has a nonzero number of links. This termination criterion is more relevant and natural in practice. In the case of the ROCK algorithm the user must specify the number of clusters $k$ and he should have an idea of the similarity threshold $\theta$. In QROCK, it is justified that specifying the similarity threshold is more practical than specifying the desired number of clusters a priori.

The authors present a very efficient method for obtaining the final clusters of the ROCK algorithm when these have no links between one another, by using the following primitives of the abstract data type MFSET [3].

1. $\text{merge}(A, B)$: takes the union of the components (clusters) $A$ and $B$.

2. $\text{find}(x)$: returns the component of which $x$ is a member.

3. $\text{initial}(x)$: creates a component that contains only the element $x$.

Algorithm QROCK:

Input: A set $D$ of data points

\begin{verbatim}
begin
    compute \text{nbrlist}[i] for each $i \in D$ using $\theta$
    for each $x$ in $D$ \text{initial}(x)
    for each $i$ in $D$
        Take a fixed point $x$ in \text{nbrlist}[i]
        for each other $y$ in \text{nbrlist}[i]
            $A = \text{find}(x)$
\end{verbatim}
\[
B = \text{find}(y) \\
\text{if } A \neq B \\
\quad \text{then } \text{merge}(A, B) \\
\}
\]

5.2.3 Proposed Phase II

The input of this phase are the small clusters obtained in Phase I. Each cluster has a unique representative as described in Section 4.2.1 in Chapter 4. We redefine the concept of neighbors and links between pairs of these small clusters as given below.

Let be \( C \) the set of small clusters obtained in Phase I and \( D \) be the given dataset. Given a minimum similarity threshold \( \theta \) a document \( d \in D \) is said to be a neighbour of \( c \in C \) if \( \text{sim}(d, c) \geq \theta \) where the \( \text{sim}(d, c) \) function is as defined in Section 4.2.3.

The link between two clusters \( c_1, c_2 \in C \) is the number of common neighbours between \( c_1 \) and \( c_2 \). Since we want to merge the small clusters, the similarity threshold used in Phase II is kept higher that the value used in Phase I.

The second phase of the proposed two-phase clustering algorithm is formally described in the following algorithm. For every document \( d \) in \( D \), we find all clusters \( x \) in \( C \) that are neighbors of \( d \) according to a user specified threshold \( \theta \). These set of clusters which are neighbors of \( d \) will form \( \text{nbrlist}[d] \). Then we can simply take a cluster \( x \) and merge the components of all other clusters of \( \text{nbrlist}[d] \) with the component of \( x \). Any two clusters in the \( \text{nbrlist}[d] \) will have \( d \) as their common neighbor.
The algorithm accepts as input the following:

- The set $D$ of data points (i.e. preprocessed documents)
- The set $C$ of cluster representatives from Phase I
- The value of $\theta$ which is the threshold used to determine the neighbors

```
begin
  for each $x$ in $C$ initial($x$)
  for each $d$ in $D$
  {
    compute nbrlist[$d$] for each $d \in D$ using $\theta$
    Take a fixed point $x$ in nbrlist[$d$]
    for each other $y$ in nbrlist[$d$]
    {
      $A = find(x)$
      $B = find(y)$ begin
      if $A \neq B$
        then merge($A, B$)
    }
    delete nbrlist[$d$]
  }
end
```

In the algorithm, the functions $initial(x)$, $A = find(x)$ and $merge(A, B)$ are as described in Section 5.2.2.

5.2.4 Complexity of Phase II

Let $n$ be the total size of the input dataset, $m$ be the size of the feature set, and $l$ (i.e., the size of $C$) be the number of small clusters obtained in phase I of the
proposed two-phase algorithm. The complexity of computing the similarity value between a pair of cluster is \( O(m) \). For each \( d \in D \), \( \text{nbrlist}[d] \) is found. So the complexity for computation of \( \text{nbrlist}[d] \) for each \( d \in D \) is \( O(lmn) \). The cost associated with the statement "for each \( x \) in \( C_{\text{initial}}(x) \)," is \( O(l) \). Since each \text{merge} \ and \text{find} \ operation could be done in \( O(1) \) time, suppose the average number of neighbours for \( d \) (i.e., the average size of \( \text{nbrlist}[d] \)) is \( p \). Then the merging process will contribute \( O(np^2) \) to the overall complexity. Thus the overall complexity of phase II is \( O(l + lmn + np^2) = O(lmn + np^2) = O(lmn + nl^2) \), since \( p < l \).

5.3 Experimental Result

The proposed algorithm was implemented and tested on following datasets:

I. BankSearch: We used the subsets ADJ, ABC and ADGHJ

II. 20NewsGroup: We used the subsets A2 and B2

III. Reuters-21578: We used a subset consisting of five categories

We also implemented QROCK algorithm and tested on the same datasets. Experiments were conducted using the fuzzy based similarity measure and cosine measure. The clustering results were evaluated using rand index [21]. The experimental results of the proposed agglomerative algorithm, two-phase clustering technique and QROCK are reported in Table 5.1.
Table 5.1 Comparison of the performances of proposed algorithms (Phase I only and Two-Phase) and QROCK

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Proposed Algorithm (Phase I only)</th>
<th>Proposed Algorithm (Two-Phase)</th>
<th>QROCK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cosine Measure</td>
<td>Fuzzy Similarity Measure</td>
<td>Cosine Measure</td>
</tr>
<tr>
<td>ADJ</td>
<td>0.86221</td>
<td>0.80213</td>
<td>0.87959</td>
</tr>
<tr>
<td>ABC</td>
<td>0.68574</td>
<td>0.73207</td>
<td>0.79604</td>
</tr>
<tr>
<td>ADGHJ</td>
<td>0.81798</td>
<td>0.80593</td>
<td>0.82025</td>
</tr>
<tr>
<td>A2</td>
<td>0.75582</td>
<td>0.76484</td>
<td>0.77489</td>
</tr>
<tr>
<td>B2</td>
<td>0.71836</td>
<td>0.77489</td>
<td>0.74721</td>
</tr>
<tr>
<td>Reuters</td>
<td>0.723577</td>
<td>0.803382</td>
<td>0.803684</td>
</tr>
</tbody>
</table>
5.4 Conclusion

In this chapter we proposed a two-phase document clustering approach. In the initial phase, we apply the clustering algorithm proposed in the previous chapter to the dataset. In the next phase, the clusters obtained from the previous phase are merged based on the concept of neighbors. This allows us to obtain clusters of arbitrary shapes. In the first phase, the clustering result was dependent on the input order of the data point. This drawback is removed in the second phase. Also outlier is removed in this phase.