CHAPTER 2

SIMILARITY BASED CLUSTERING

2.1 INTRODUCTION

Clustering is a simple mechanism to group similar elements. It can be considered as an unsupervised learning problem. A cluster is therefore a collection of elements, which are “similar” between them in the same cluster and are “dissimilar” to the elements belonging to other clusters. Elements within the same cluster have more similar properties. Clustering is an intelligent technique for mining data. It has been utilized as an excellent way of grouping data in databases or documents by their content or structure. Similar elements are moved into closer places. It is quite natural that similar elements may be accessed for computing summarization or manipulated as a group. It is difficult for the system to identify the similarity between the elements directly. It is not possible to build the system to automatically detect the similarity between elements. Figure 2.1 shows the clustering principle with a graphic example. Based on the distance between the elements present, the elements are divided into four groups. Two or more elements belong to the same cluster if they are “close” according to a given distance. The major applications of clustering are

- Marketing
- Biology
- Libraries
The goal of clustering is to determine the intrinsic grouping in a set of unlabeled data. To decide about good clustering is a complex task. There is no absolute best criterion, which would be independent of the final aim of the clustering. Consequently, it is the users who must supply this criterion, in such a way that the result of the clustering will suit their needs.

The main requirements that a clustering algorithm should satisfy are:

- scalability;
- dealing with different types of attributes;
- discovering clusters with arbitrary shape;
- minimal requirements for domain knowledge to determine input parameters;
- ability to deal with noise and outliers;
- insensitivity to order of input records;
- high dimensionality;
- interpretability and usability.

The following problems are associated with clustering and current clustering techniques do not address all the requirements adequately and concurrently.

- dealing with a large number of dimensions and large number of data items can be problematic because of time complexity;
- the effectiveness of the method depends on the definition of “distance” (for distance-based clustering);
- if an obvious distance measure doesn’t exist it must be defined, which is not always easy, especially in multi-dimensional spaces;
- the result of the clustering algorithm (that in many cases can be arbitrary itself) can be interpreted in different ways.

### 2.2 DATA PREPROCESSING

Data preprocessing is an important step in the data mining process. Real world data are generally incomplete, noisy and inconsistency. The garbage data should be removed from the collected, to improve the quality of the data. It is part of data mining and machine learning process. Data collection methods are often loosely controlled, resulting in out-of-range
values, impossible data combinations, missing values, etc. Analyzing data that has not been carefully checked for such problems can produce misleading results. Thus, the representation and quality of data are of first and foremost importance before running an analysis.

The major tasks in data preprocessing are

- Data cleaning
- Data integration
- Data transformation
- Data reduction
- Data discretization

Data cleaning is a process of:

- Filling in missing values
- Smooth noisy data
- Identify or remove outliers
- Resolve inconsistencies.

The different approaches used to fill in missing values are

- Ignore the tuple.
- Use the attribute mean.
- Predict the missing value by using a learning algorithm. The methods used to identify the outlier and smooth out noisy data are

- Binning
  
  Sort the attribute values and partition them into bins
- Clustering
  Group values in clusters and then detect and remove outliers

- Regression
  Smoothen by fitting the data into regression functions.

Inconsistency present in the data can be removed by using domain knowledge or expert guidance.

Data transformation is a process of performing normalization or computing aggregation or generalization. Normalization is a process of scaling attribute values to fall within a specified range.

Example:
To transform V in \([\text{min}, \text{max}]\) to \(V'\) in \([0,1]\), apply \(V'=(V-\text{Min})/(\text{Max}-\text{Min})\)

Scaling by using the mean and standard deviation (useful when min and max are unknown or when there are outliers): \(V'=(V-\text{Mean})/\text{Sdev}\).

Aggregation is moving up in the concept hierarchy on numeric attributes. Generalization is moving up in the concept hierarchy on nominal attributes. Attribute construction: replacing or adding new attributes inferred by existing attributes.

The data reduction mechanism is used to produce the reduced volume of data yielding the same analytical result. The different approaches used for data reduction are

- Reduce the number of attributes
- Reduce the number of attribute values
- Reduce the number of tuples.
Data discretization is a part of data reduction, replacing numerical attributes with nominal ones. The types of discretization are

- Unsupervised discretization
- Supervised discretization

In unsupervised discretization equal-interval (equiwidth) binning is used to split the whole range of numbers in intervals with equal size or Equal-frequency (equidepth) binning is supported with the use intervals containing equal number of values.

Supervised discretization uses the values of the class variable and class boundaries. The three step procedure in this method is

- Sort values.
- Place breakpoints between values belonging to different classes.
- If too many intervals, merge intervals with equal or similar class distributions.

### 2.3 CLASSIFICATION OF CLUSTERING ALGORITHMS

Clustering algorithms can be classified as

- Exclusive Clustering
- Overlapping Clustering
- Hierarchical Clustering
- Probabilistic Clustering

In the exclusive clustering data are grouped in an exclusive way, so that if a certain datum belongs to a definite cluster then it could not be
included in another cluster. In overlapping cluster, the data may belong to two or more clusters. The results of the hierarchical clustering are the union between the two nearest clusters. Probabilistic approach is followed in last clustering method.

Based on the classification strategy the mainly used clustering algorithms are

- K-means
- Fuzzy C-means
- Hierarchical clustering
- Mixture of Gaussians

### 2.3.1 K-Means Clustering Algorithm

It belongs to the category of exclusive clustering. An important criterion used in clustering algorithm is the distance measure between data points. If all the data units belong to the same physical units, then the simple Euclidean distance metric is adequate to group the data. Sometimes different scaling parameters specified for the data unit led to the data units belonging to different clusters. This is due to the mathematical formula used to combine the distances between the single components of the data feature vectors into a unique distance measure that can be used for clustering purposes: different formula are lead to different clustering. This is illustrated in Figure 2.2.

The domain knowledge is required to guide the formulation of a suitable distance measure for each particular application. A popular measure called Minkowski is used for higher dimensional data. This algorithm is one
of the simplest unsupervised learning algorithms that solve the well known clustering problems. Initially k clusters are specified in which the procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed initially.

Figure 2.2 Effect of scaling in clustering

The basic idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because different locations cause different results. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point it is need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop, user may notice that the k centroids change their location step by step until no more changes are done. Although it can be proved that the procedure will always terminate, the k-means algorithm has not necessarily found the most optimal configuration, corresponding to the global objective function minimum. The algorithm is
also significantly sensitive to the initial randomly selected cluster centers. The k-means algorithm is executed multiple times to reduce this effect.

2.3.2 Fuzzy C-means Clustering Algorithm

This clustering algorithm is a method of overlapping of clustering, which allows a single piece of data to belong to two or more clusters. It is proposed by Dunn in 1973 and improved by Bezdek in 1981 and frequently used in pattern recognition.

2.3.3 Hierarchical Clustering Algorithm

The basic steps of a hierarchical clustering algorithm are

i. Given N data items are used to create N Clusters.
ii. Identify the closest pair of clusters and combine to form a single cluster.
iii. Calculate the distance between old and new cluster.
iv. Repeat the steps (ii) and (iii) until all elements are clustered into a single cluster of size N

2.3.4 Mixture of Gaussians

Clustering can be done by using model based algorithm also. It consists of certain models of clusters and attempting to optimize the fit between the data and the model. Each cluster can be mathematically represented by a parametric distribution, like a Gaussian (continuous) or a Poisson (discrete). The entire element set is modeled by a mixture of these distributions. An individual distribution used to model a specific cluster is often referred to as a component distribution. The merits of this approach are
• Mathematical proofs based inference techniques available;
• flexibility in choosing the component distribution;
• obtain a density estimation for each cluster;
• support soft classification

The most popular method of this category is the one based on learning a mixture of Gaussians.

2.4 XML DATABASES

Data used in internet applications and commercial business applications can be either structured or semi-structured or unstructured. Structured data can be easily represented in either traditional Relational Database Management Systems (RDBMS) or Object Oriented Database Management Systems (OOSDBMS) or File Systems. It is difficult to manage unstructured or semistructured data. The data which does not have a rigid structure or structure can be frequently changed is called semistructured data. Separate tools or generalized software are required to manage unstructured or semi-structured data. Most of the database management systems used platform dependent and proprietary software. XML becomes a standard data format for the management, display and organization of data in a platform and language independent way to represent both unstructured and semistructured data. Initially it was not designed for database applications.

The World Wide Web (WWW) is based on Hyper Text Markup Language (HTML). In HTML each tag has its predefined meaning and its purpose. The syntax for XML is derived from hypertext markup language and Standard Generalized Markup Language (SGML). User can define his own tag in XML for data descriptions. XML is an essential technology for the user
working with data whether on the web or internally. It is a technology concerned with the description and structuring of data. Representing data in XML is inefficient because the values of elements are stored along with tags. Tags occupy a portion of storage space.

The major benefits of XML are.

- Self description of data
- Format is not rigid
- It supports nested structures
- It is widely accepted
- Portable
- Good for storing tree structure or graph structure data
- Adaptable to change

Compared to the conventional database system XML has the following limitations

- Efficient access to data due to parsing and conversion
- Efficient Storage
- Indexes
- Security
- Multi-user access
- Triggers
- Transaction Management
- Data Integrity
- Queries across multiple documents
2.5 SIMILARITY BASED CLUSTERING ALGORITHM

The XML data available in different sources and in different format must be transformed into uniform format during data integration. After collecting data from different sources noises, inconsistencies are removed by using the preprocessing procedure presented in the section 2.2.

The basis for similarity based clustering algorithm is defined with the same type of elements present in the XML document. In this method, our human expertise is used to identify and specify the similar elements present in the document. The entire XML data and schema of the data are displayed to the user. Based on his expertise, a user can specify the element which is used to cluster the entire XML document into clusters. Similar elements present in the XML documents are grouped. The number of clusters is computed based on the different data values of an XML element.

Similarity based algorithm is implemented as two types. In the first version, it computes the number of clusters to be created initially by reading the entire input XML data file based on the distinct value present in the XML element. After computing the specified number of clusters to be created, the input file contents are read once again and the elements are moved to the corresponding cluster based on the element value. After clustering, all the data present in different clusters are merged into form a single XML data file.

In the second version of the algorithm, the clusters are created dynamically by reading the input file content only once. Initially a single cluster is created and the first element is moved into an initial cluster and the element value is stored in a list for further operation. For the successive element, the element values are matched with the value present in the list. If
a match occurs then the element is moved to the corresponding cluster, otherwise a new cluster is created and the current element is moved into the newly created cluster. This operation is repeated for all the elements present in the file.

2.5.1 Architecture Model

The architecture diagram is shown in the Figure 2.3. The data stored in the repository are handled by the Berkeley database engine interface. This database engine directly interacts with the repository to retrieve the XML data in the container.

![Figure 2.3 Clustering system architecture](image)

The data along with schema is retrieved and schema is displayed to the user. The user specifies the XML element as an input parameter. Based on
the user input, the elements are clustered. The Figure 2.4 and 2.5 shows the hierarchical representation of XML data before and after clustering.

Figure 2.4 Hierarchical representation of data before clustering

Figure 2.5 Hierarchical representation of data after clustering
2.5.2 Algorithm

Clustering algorithm //Two pass and merge operation) Version 1

Name: Cluster_Similarity

Input : SC, XD
SC Schema of the input data.
XD xml data file

Output CD clustered data

Algorithm steps:

Step 1: input SC,XD

Step 2: If schema not null
User scans the entire schema
Scan(SC)

Step 3: User scan the entire XML data
Scan(XD)

Step 4: User specify the parameter based on which clustering to be done
Input param

Step 5: Compute the number of clusters to be created
i. Initialize count to 0
ii. For each element in the XD repeat the step
iii. Read param value from XD;
iv. If param value is distinct
   count=count +1

Step 6: create cluster(CD,count)
Based on the count value create CD clusters.

Step 7: Move the element in XD to the corresponding clusters
   i. For each element in the XD repeat the step ii and iii
   ii Read each element from XD
   iii. Move an element to the corresponding cluster move
        (element,CD)

Step 8: Merge the clustered data (CD)

Step 9: Return the clustered data(CD)

Clustering algorithm (Single Pass and Merge operation version 2)

Name: Cluster_Similarity

Input : SC, XD
   SC Schema of the input data.
   XD xml data file

Output: number of clusters and the clustered data

Algorithm steps:

Step 1: input SC,XD

Step 2: If schema not null
   User scans the entire schema
   Scan(SC)

Step 3: User scan the entire XML data
   Scan(XD)

Step 4: User specify the parameter based on which clustering to be done
   Input param
Step 5: Create initial cluster and initialize count to 0
Create a list paramlist to store the distinct param value.

Step 6: Repeat the step I and II until all the elements in the input XML file are processed.

i. Read the element from the input file and access the param value.

ii. If param value exists in the paramlist then
    Move the element to the corresponding cluster
Else
    Create a new cluster
    Move the current element to the new cluster
    count=count +1

Step 7: Merge the clustered data to form a single XML data file

Step 8: Return the clustered data file.

2.5.3 Experimental Setup

Similarity based clustering algorithm was implemented in C and all experiments were run on a PC with a 2.66 GHz Intel core 2 Duo processor, 250 GB SATA HDD and 2GB DDR RAM with windows XP operating system environment. The SQLserver2005 and the open source software tool Berkeley DB XML are used as database engines to store and manipulate XML data. The similarity based clustering algorithm implemented is used to cluster the XML data stored in the container of Berkeley DBXML database engine and Microsoft SQL server2005 database engine.

One major advantage of the modern native XML databases is their ability to index the XML documents they contain. Proper use of indices can
significantly reduce the time required to execute a particular Xquery expression. If no index exists, then SQLServer2005 evaluates each and every element in the table against the query. If the index exists, SQLServer2005 can find a subset of matching documents with a single or significantly reduced set of lookups. By carefully applying XML indexing strategies, the retrieval performance can be improved considerably. Indices are specified in four parts: path type, node type, key type and uniqueness. Figure 2.6 shows the time required to access the elements of an XML document stored in a table using an index and without indexes. Table 2.1 contains the access time for the retrieval of data from the table in SQLserver2005 and Berkeley DBXML database engine. It shows that indexing reduces the time required to retrieve elements of the XML document from the database.

2.5.4  SQLServer2005

The Microsoft SQLServer2005 database engine is the core service for storing, processing and securing data. The database engine provides controlled access and rapid transaction processing to meet the requirements of the most demanding data consuming applications within the enterprise.

The SQLServer2005 database engine also provides rich support for sustaining high availability. In SQLServer2005, the xml data type allows to store XML documents and fragments in SQLServer database. An XML fragment is an XML instance that is missing a single top-level element. In a table, XML data type can be used to create columns and variables that store XML instances in them. It is to be noted that the stored representation of XML data type instances cannot exceed 2GB. XML instances are stored in the XML type columns as BLOBs or CLOBs. These XML instances can be large and stored in binary representation. Without an index, these binary large objects are shredded at run time to evaluate a query that can be time
Indices can reduce the amount of data that must be read to return the query result set.

Figure 2.6 Effect of indexing in data retrieval

Table 2.1 Effect of indexing in data retrieval

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Number of elements retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.5k</td>
</tr>
<tr>
<td>Without index BDXML</td>
<td>2.31</td>
</tr>
<tr>
<td>Without index SQL</td>
<td>0.33</td>
</tr>
<tr>
<td>With Index BDXML</td>
<td>0.625</td>
</tr>
<tr>
<td>With index SQL</td>
<td>0.314</td>
</tr>
</tbody>
</table>
The XML indices fall into two categories: Primary XML index, Secondary XML index. The first index on the XML type column must be the primary XML index. Using the primary XML index, three types of secondary indices are supported. These include PATH, VALUE and PROPERTY. Depending on the type of queries, these secondary indices may help improve the query performance. Figure 2.7 shows the query for the generation of sample XML document in SQL Server 2005.

2.5.5 Berkeley Database XML

Berkeley Database XML (BDBXML) is an embedded XML database engine that provides support for Xquery access. In Berkeley DBXML all XML data are stored within files called containers. The Berkeley DBXML shell provides a simple and convenient way to work these containers and exposes most of the Berkeley DBXML functionality in a friendly, interactive environment. Containers also store XML documents as either whole documents or as nodes. When containers store the whole documents, all the XML documents are stored as one unit in the containers exactly as it was presented to the system. When documents are stored as nodes, the XML document is deconstructed into smaller pieces-nodes and those small chunks

```sql
create table empc([Sno] [int] not null,[Empinfo] [xml] not null);
create unique index ui_emp on empc(Sno);
create fulltext catalog ft as default;
create fulltext index on empc(Empinfo) key index ui_emp;
Insert into empc (Sno, Empinfo) values (1,XMLdata)
```

**Figure 2.7 Data generation SQL query**
are stored in the container. Node storage offer better performance than
document storage and for this reason node storage is the default container
type.

2.6 PERFORMANCE ANALYSIS

The performance of similarity based clustering algorithm is
analyzed in the database environment SQLserver2005 and Oracle Berkeley
DBXML. The experiment is conducted by creating a suitable database table
and inserting the sample data. Select query in the SQL server2005 is used to
retrieve the data from the database with out and with full-text search index for
the unclustered data. The data retrieval operation is also repeated for the
clustered data.

2.6.1 SQLServer2005 Performance

The similarity based clustering algorithm is tested using several
thousand XML elements stored in a SQL Server database. Initially the
database named empfull.mdb is created. In this database, table named empc is
created with the following fields Sno (int) and Empinfo (xml). The XML
column Empinfo contains name, department, designation and address as
elements. Ten thousand well formed XML elements are inserted into the
table. From the table, queries can be used to retrieve the data. For example,
without indexing and clustering, the following queries were used to retrieve
data. Timestamp before and after the query command present the time
required to retrieve the data from the table. The following Xquery would read
as “from the table named empc select all staff elements that contain
designation as ‘System Analyst’ ”
select current_timestamp

select Empinfo.query('/staff/[desig="System Analyst"]') from empc

select current_timestamp

The query “from the table named empc select all staff elements that contain the designation as ‘System Analyst’ or ‘Project Manager’”

select current_timestamp

select empinfo.query('/staff/[desig="System Analyst" or desig="Project Manager"]') from empc

select current_timestamp

The time required for executing the query without indexing and clustering is 0.58 seconds. One of the major advantages of modern native XML databases is their ability to index the XML elements they contain. Proper use of indices can significantly reduce the time required to execute a particular Xquery expression. The column Empinfo elements are retrieved and clustered using the designation field. The elements with designation value “System Analyst” are grouped and placed at the beginning of the document and similarly other groups are placed. After clustering, the clustered data are inserted into the table empc and is indexed with column empinfo. Now the queries are executed to retrieve the elements of the particular category.

The performance comparison graph in Figure 2.8 shows the time required to retrieve XML data from the table under different strategies. The y-axis represents the time required to retrieve XML data from the table in seconds. X-axis represents the number of elements retrieved. The time required to retrieve 2500 elements of a particular type from the table without cluster and indexing is 0.33 seconds.
After indexing, time required for retrieving the data is 0.314 seconds. The time required to retrieve the same data with cluster and indexing is 0.19 seconds. The data retrieval time for SQLServer2005 under different categories such as without cluster and index, with cluster alone, with index alone and cluster with index are stored in a Table 2.2. The experimental result show that XML data from the SQLserver can be read efficiently by using clustering with indexing mechanism.

2.6.2 Berkeley DBXML Performance

In BDBXML the XML data are maintained in the containers. Initially container named staff.dbxml is created. In this container ten thousand elements are inserted in the following field such as serial number as attribute, name, department, designation, address as elements. The command create container is used for creating the container and elements are inserted into the container by putDocument.

Table 2.2 Performance comparisons in SQLserver2005

<table>
<thead>
<tr>
<th>Strategy</th>
<th>2.5k</th>
<th>5.0k</th>
<th>7.5k</th>
<th>10k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without cluster and index</td>
<td>0.33</td>
<td>0.58</td>
<td>0.767</td>
<td>0.85</td>
</tr>
<tr>
<td>Cluster alone</td>
<td>0.31</td>
<td>0.5</td>
<td>0.67</td>
<td>0.84</td>
</tr>
<tr>
<td>Index alone</td>
<td>0.314</td>
<td>0.53</td>
<td>0.7</td>
<td>0.84</td>
</tr>
<tr>
<td>Cluster with index</td>
<td>0.19</td>
<td>0.34</td>
<td>0.47</td>
<td>0.51</td>
</tr>
</tbody>
</table>
Figure 2.8 Performance comparisons in SQLserver2005

In the Berkeley DBXML shell prompt, user can enter the commands. The shell prompt is displayed as

dbxml> create container staff.dbxml
creating node storage container with nodes indexed

dbxml>putDocument "" ""

for $i in ( 0 to 2499)
return

<staff sno="{$i}">
  <name> - - -</name>
  <department> - - - </department>
  <designation>System Analyst</designation>
  <address> - - -</address>
</staff>
This putDocument command is used to insert ten thousand elements to the document named staff.dbxml. Index for the document is created with addindex command. Indices are specified in four parts: path type, node type, key type and uniqueness. The query requires an index of the node elements to determine if something is present or not. Since the pattern is not expected to be unique, then it is not necessary to turn on uniqueness. The Berkeley DBXML index type used is node-element-presence-none.

```
dbxml>addIndex """" designation node-element-presence-none
```
Adding index type: node-element-presence-none to node :{}: designation

From the container, staff.dbxml queries can be used to retrieve the data. For example, without indexing and clustering the query is used to retrieve data. The query could be read as “from the container named staffdbxml select all staff elements that contain the designation as “System Analyst”.

Time with query command displays the output of the query with time taken to complete the execution of the query.

dbxml> time query
‘collection("staff.dbxml")/staff/name[designation="System Analyst"]’

Time in seconds for command query is 2.313.

This time is required for retrieving the data from the container without indexing and clustering. With indices, Berkely DBXML can find the subset of the matching elements with a single or significantly reduced set of lookups. By carefully applying indexing strategies the retrieval performance can be improved considerably. The container elements are first clustered using field designation. After clustering, the clustered data are inserted into the container staff.dbxml and is indexed with field designation. Now the queries are executed to retrieve the elements of the particular category.

The performance comparison graph Figure 2.9 shows the time required to retrieve XML data from the container under different strategies such as without cluster and index, with index alone, with cluster alone and cluster with index. The y-axis represents the time required to retrieve XML data from the whole document container in seconds.
Figure 2.9 Performance comparisons in Berkely DBXML

X-axis represents the number of elements retrieved. The time required to retrieve 2500 elements of a particular type, from the container without clustering and indexing is 2.313 seconds. After indexing, time required for retrieving the data is 0.625 seconds. The time required to retrieve the same data with clustering and indexing is 0.462 seconds. This graph proves that, clustering with the indexing procedure yields better performance.

Table 2.3 contains the time required for retrieval of data from the container of Berkely DBXML database. Clustering increases performance for data retrieval to a certain extent. Indexing supports fast retrieval of the data for manipulation. The experimental results indicated that the proposed similarity based clustering method with indexing provides better performance in data retrieval.
Table 2.3 Performance comparisons in Berkeley DBXML

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Number of elements retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.5k</td>
</tr>
<tr>
<td>Without cluster and index</td>
<td>2.31</td>
</tr>
<tr>
<td>Cluster alone</td>
<td>1.94</td>
</tr>
<tr>
<td>Index alone</td>
<td>0.625</td>
</tr>
<tr>
<td>Cluster with index</td>
<td>0.462</td>
</tr>
</tbody>
</table>

2.7 CONCLUSION

Performance of similarity based clustering method with indexing is analyzed for XML document in SQLServer2005 and Berkeley DBXML. From the results it is concluded that the indexing alone does not yield the expected performance improvement. When clustering is combined with indexing it offers better performance than the expected level of indexing or clustering alone. The existing XML indexing techniques are used only to retrieve the data exclusively from the Berkeley DBXML database engine or Microsoft SQLServer2005 database engine and not from the flat file.

The next chapter describes in detail the proposed indexing model. The algorithm proposed are also presented in this chapter along with results obtained.