CHAPTER 5

DIS-SIMILARITY CALCULATION USING GROUPING

5.1 INTRODUCTION

This chapter presents a grouping induction algorithm for duplicate detection. It combines the merits and features of clustering and classification algorithm to generate group for reducing the number of unwanted comparison, while calculating the dis-similarity value. The performance of the algorithm has been evaluated and compared with the existing algorithm.

Work on detecting related groups of entities from link information also spreads over many different databases. Extensive work has been done in the area of database information retrieval, where the task is to cluster database with regard to information. Each database is observed as a bag of words, and information having similar distributions over words is considered similar. Another approach involves probabilistic modelling of the semantic space of documents with latent variables Hofmann (1999), Blei (2003). Each cluster is modelled as a mixture of 'topics' or meaningful word distributions. It is not clear if complex 'group mixture' models are required for explaining relations of parent cluster into child cluster and so on. Several researchers have made use of entity relationships for iteratively classifying or categorizing entities. Taskar et al (2001) have extended the framework of probabilistic relational models to propose a joint model for classification and clustering for relational data. Recently, work has been done in probabilistic modelling of stochastic links given groups of entities Kubica et al (2003 a,b), Kubica et al (2002). Their
models however do not consider uncertainty in the identity of entities. Also, while they look to learn the generative model directly using stochastic gradient ascent techniques, ours is a clustering approach that uses the probability of the observed links given the generative model to define a clusters group. This is similar in flavour to information theoretic distance measures that have previously been used for clustering Dhillon et al (2003).

5.2 DE-DUPLICATION OF ENTITIES: FORMULATION

In the de-duplication problem, we have some collection of cluster to entities and from this set of cluster we would like to identify the (unique, minimal) collection of individuals or entities to which they should be mapped. In other words, we would like to find a many-to-one mapping from references to entities. But a priori we do not know the set of entities.

Formally, we have a collection of references $R = \{r_1, r_2, \ldots, r_n\}$. Each reference $r$ corresponds to a unique entity, $E(r) \in \{e_1, e_2, \ldots, e_k\}$. The references are members of links $L = \{l_1, l_2, \ldots, l_m\}$. Each link is a collection of reference and a reference appears in only one link. In our citation example, each observed name is a reference, the list of names from database form a link and the hidden entities belong to name entities. Given $R$ and $L$, our task is to correctly determine both entities (including the number of entities $k$) and the mapping from references to entities.

Not surprisingly, we do this by clustering references that are similar to each other. The key to the success of this clustering algorithm is the dis-similarity measure (or, equivalently, the distance measure). We define a distance measure that takes into account both the attributes of the entities and the links between them and finally rank it.
To compute the similarity of the relationships that two entities participate in record, we do not want to compare their links. But if we simply compare whether the references are the same link if required, then we will not find any overlap references that are derived from the name and, as we defined them, each cluster reference is distinct.

5.3 LINK MEASURE FOR DE-DUPLICATION

Our aim is to construct the relation $\text{dup}(ri, r3)$ over the set of names retrieved from the database. This relation is symmetric and also reflexive, meaning that every reference is its own duplicate. Our definition of duplicates is based on a measure between given names.

$$\text{dup}(ri, r j) = \text{true if } d(r , r j) < t$$

for a given threshold $t$ and false otherwise. This distance measure is a weighted combination of the attribute of the references. We define the measure in the following subsection. The definition of duplicates is recursive in that the distance between references is tied to the current set of duplicates.

The distance between two references is defined as

$$d(ri, r) = (1 - \theta) \cdot X \cdot \text{dattr}(ri, rj) + c \cdot \text{dlink}(\text{L}(ri), \text{L}(rj))$$

where $\text{dattr}$ $\theta$ is the distance between the attributes, $\text{dunk}()$ is the distance measure between link sets of references and $\alpha$ is a weighting of the two distances.

We can now see the recursion in the definition of duplicates and appreciate the need for an iterative algorithm. As new duplicates detection algorithm are discovered, the distances between link sets of cluster are going to change, potentially leading to the discovery of more duplicates. We represent
the current sets of duplicates as clusters. We associate with each cluster the set of links that its references occur in. This is the link set of the cluster. Formally, for a cluster $C_k$,

$$L(C_k) = \{/(\text{iri})\text{ E CO.}\}$$

Also, with each cluster, we maintain a representative attribute value of all its clusters. At each step, the algorithm re-evaluates the clusters and merges the 'nearest' cluster-pair to represent the same entity. The iterations continue till there are no more name worthy of merging.

### 5.4 GROUP DETECTION: FORMULATION

Any algorithm that directly clusters the entities must allow overlap among the clusters, so that an entity may belong to multiple groups. Most clustering algorithms do not have this feature. However, we can make use of the fact that the entities in a link come from the same group. Instead of clustering the entities, we can cluster the links, so that the links that come from the same group are clustered together. Here we are looking for a partition of the links into clusters and any clustering algorithm with an appropriately defined distance measure should work. Each link cluster will now represent a group that we are interested in and an assignment of entities to groups follows naturally. Each cluster/group $q_i$ has a set of links $L(q_i)$. The set of entities assigned to the group will be

$$E(q_i) = \{E \text{ L}(q_i)\}$$

It may be noticed that we are not constraining any entity that belongs to a single group. An entity that occurs in two different links A and B that have different group labels $q_1$ and $q_2$ will be assigned to both group’s $q_1$ and $q_2$. 
Each cluster or group is associated with its own sub-set of its cluster and it is meant as child cluster.

5.5 PROPOSED GROUPING ALGORITHM FOR DUPLICATE DETECTION

It is based on sequential covering approach. First generate the parent list and then add its attributes to continuous attribute child list. Incorporate the Alternative Decision Tree (ADTree) to handle splitting of bunch of record into self-sized group.

Spectral ADTree approaches aim to identify lower dimensional sub-spaces (Eigen vectors) from the database information and cluster information with similar components along these sub-spaces cluster. The basic idea is again to identify the principal vectors for different forms of the information of all data bases and this is done with an iterative manner. The concept has also been augmented to have a probabilistic interpretation Cohn (2000) and combined with information models to propose a unified model for detecting duplicate within the each group of information retrieve from database. The data warehouse domain is inherently directional, and it distinguishes it from our motivation. Our focus is also to use the links for calculating dis-similarities percentage among each group and ranking them with regard to importance or relevance.

An Alternative Decision Tree (ADTree) is a learning Bansal et al (2004) method for record classification. It generalizes decision tree for performing the supervised learning. The goal is to create a model that predicts the value of a target variable based on several input variables. In our scenario prediction should be short name (Xi) with respect to the full name (Yi). It consists of two nodes i.e decision nodes and prediction node. Decision nodes
specify the prediction condition, if condition is satisfied return TRUE else FALSE.

![Diagram](image)

**Figure 5.1 Multi-Level Clustering Formation**

Consider the Training dataset \{(X_1, Y_1) \ldots (X_i, Y_i) \ldots (X_n, Y_n)\}, each X is an example with a label Y. A set of weights \(W_i\) corresponding to each instance node.

**Procedure ADTree_technique()**

```
{
if (is_valid[short_name])
then
    if (contains_cluster[short_name])
then
        return existing_cluster[short_name];
else
    return new_cluster[short_name];
endif
else
```
MLC forms a tree for the clustering process (Figure 5.1). In the tree structure, the height of each level of nodes represents the similar degree among existing hierarchical clustering problem. Here we use any split algorithm for splitting data into a cluster; alternatively we are using ATree technique for splitting a whole data into cluster. ATree divides the data into short name, if cluster is already available with the short name then insert a record into same cluster or else create a new cluster with the new name of short name and then insert into a new cluster. In each clusters sub-set short names pointing to the whole records are created.
Procedure ADTree_groupInductionAlgorithm ()
{
Initialize: Parent_List[n] ⇐ 0,
         Child_List[n] ⇐ 0,
         Grand_child_list[n]⇐0;
Loop L1: while !endOfRecord[Record]
         C1 ⇐ Level_1_attr_value;
         Position ⇐ Size[Parent_List]+1;
         If(is_valid[ C1])
         then
          C2 ⇐ Level_2_attr_value;
          C3 ⇐ Level_3_attr_value;
          Child_Position ⇐ Size[Child_List]+1;
          Loop L2: while !endOfRecord[Record]
                     If(!contains[parent_list, C2])
                     then
                      Parent_List[Position]⇐Create new Cluster C1,
                      Parent_Position;
                      Child_List[n] ⇐ Create new Cluster C2,
                      Child_Position;
                      Grand_child_list[n]⇐Create new Cluster C3,
                      Child_Position;
                      Parent_List[Position]⇐Insert into new Cluster
                      {Parent_Position,
                      (C1,vector[Record Information])};
                      Child_List[n] ⇐ Insert into new Cluster
                      {Child_Position,
                      (C2,vector[Record Information])};
Grand_child_list[n] \leftarrow \text{Insert into new Cluster} \\
\{\text{Child\_Position}, \\
\text{\(C2,\text{vector[Record\_Information]}\)}\}; \\
\text{Return new\_cluster;} \\
\text{else} \\
\text{Parent\_List[Position] \leftarrow \text{Insert into existing Cluster}} \\
\{\text{Parent\_Position,} \\
\text{\(C1,\text{vector[Record\_Information]}\)}\}; \\
\text{Child\_List[n] \leftarrow \text{Insert into existing Cluster}} \\
\{\text{Child\_Position,} \\
\text{\(C2,\text{vector[Record\_Information]}\)}\}; \\
\text{Grand\_child\_list[n] \leftarrow \text{Insert into existing Cluster}} \\
\{\text{Child\_Position,} \\
\text{\(C2,\text{vector[Record\_Information]}\)}\}; \\
\text{\textbf{Existing\_Cluster} \leftarrow \text{call Dis-Similarity Calculation}} \\
\text{\textit{Algorithm \(C1,\)} \\
\text{\textit{vector[Record\_Information]}}} \\
\text{\text{Return existing\_Cluster;}} \\
\text{endif} \\
\text{Goto L2} \\
\text{else} \\
\text{return 0;} \\
\text{endif} \\
\text{Goto L1} \\
\}
ADTree_groupInductionAlgorithm() forms a tree for the clustering process (Figure 5.2). In the tree structure, the height of each level of nodes represents the dis-similar degree between each cluster. MLC incorporates the futures of ADTree features and it overcomes the existing hierarchical clustering problem and it reduces the time consumption for duplicate detection and number of record comparisons. Here we do not use any split algorithm for splitting data into a cluster; instead we are using ADTree technique for splitting a whole data into cluster.

Figure 5.2 Group Detection of ADTree_groupInductionAlgorithm()
ADTree divides the data based on short name; if cluster is already available with the short name then insert a record into a same cluster else create a new cluster with the new name of short name and then insert into a new cluster. In each clusters sub-set short name pointing to the whole records If cluster is already available then starts the de-duplication process else create a new cluster and then exit from the process.

Procedure DS_Calculation() shows how to use the clustering algorithm to calculate dis-similarity value for de-duplication. First Procedure TT_generation () constructing a truth table for each pair of string in the each cluster.

Procedure DS_Calculation()
{
Input:
C1 ← {Short_Name}
C2 ← {Actual_name}
VEC ← {Vector [Subject_Information]}

Process:
Loop L1: While !endOfVector[Record]
Begin
C3 ← VEC [C1].getActualName ();
\( p,n \) ← Call Truth Table Construction Algorithm\n(C2,C3);
Entropy \( (p_i,n_i) \) ← -p \( \log_2(p) \) - n \( \log_2(n) \)
Gain\( _i \) ← \( \sum \) (Entropy value of child dataset) - \( \sum \) (Entropy value of total dataset) * 100
Loop L2: While !endOfVector[C1]
Begin
C4 ← VEC [C1].getDis_Sim_Score();
If(Score_Limit>= Gain_i)
If(Gain_i > C4 and Length(C3 ) > Length(C4))
Then
   VEC ← Insert into Existing Cluster C2
   And set it as “Primary”
   VEC ← Update Existing Cluster C3
   And set it as “Secondary”
Else
   VEC ← Insert into Existing Cluster C2
   And set it as “Secondary”
Endif
Goto L2
Goto L1
Return VEC

Every Boolean function can be specified as a table with the value of 0,1 and function has a “n” argument, then the total possible argument combinations are 2^n. This section is used to construct the logical representation of two different string tokens truth table. In pair of strings which has one more length Where Ci, i=1.....n represents the i’th character of column. Rj, j=1.....m represents the j’th character of row.

\[
TT (X_i, Y_j) = \begin{cases} 
0 & \text{if } X_i \neq Y_j \\
0/1 & \text{if } X_i = Y_j \text{ and } X_i \text{ contains 1} \\
1 & \text{if } X_i = Y_j 
\end{cases}
\]
Procedure TT_generation()
{
    Initialize: String_1;
    String_2;
    Row $\leftarrow$ 0;
    Column $\leftarrow$ 0;
    Loop L1: While !empty(String_1)
        C1 $\leftarrow$ String_1[Row]
        Row = Row + 1;
    Loop L2: While !empty(String_2)
        C2 $\leftarrow$ String_2[Column]
        Column = Column + 1
        If C1 = C2
            Then
                Truth [Row, Column] = 1
            else
                Truth [Row, Column]=0
        endif
        Goto L2
    Goto L1
}

Then apply the truth table value into entropy and gain formula. The output of gain will be a dis-similarity percentage. The gain values are compared with the existing cluster gain value and if it is greater than the existing and the length of current string is greater than the existing cluster string then set current as “PRIMARY” else set the current as “SECONDARY” and its dis-similarity score.
5.6  ITERATIVE DE-DUPLICATION

We may recall that we use a combined measure for references name that takes into account the attribute as well as the link distance of references cluster. The link distance considers the current duplicates shared between two links cluster and evolves as new duplicates are detected. Accordingly, we derived an iterative algorithm. Thus, in order to jump-start our clustering algorithm, we start off by merging parent to child cluster references that 'obviously' correspond to the same cluster. Once the initial clusters have been so formed, we choose the candidate clusters that are likely to be the same entity. At each iterative step, the algorithm re-evaluates names for the candidates, selects the closest pair according to the short name measure, merges the clusters and updates the representative attributes and link to the child cluster. This procedure is continued until the end of candidate set.

5.7  EXPERIMENTAL EVALUATION

A difficulty with evaluating record linkage performance is the lack of gold standard for real-world data sets. Here we report a systematic evaluation of our algorithms on synthetic data, where we can model associations among names, quantify the amount of noise and evaluate the recall precision profiles for our algorithms.

5.7.1  Data Generator

Since one of our goals is to evaluate the importance of co-occurrence information for de-duplication, we use a unified data generator for both tasks that incorporates structure in the name domain by mimicking a real-life scenario where an account belong to which type of account, bank, region,
branch, and personal information. It creates groups of entities, where an entity belongs to a group with some probability. Since any name can be associated with multiple account groups or have multiple branch interests, we allow an entity to belong to multiple groups. Each entity has fixed attribute values, corresponding to the true identity of the name. However, when it appears as a name in any database, it is likely to look different, and that is why name identification is difficult. We mimic this phenomenon by probabilistically adding noise to the name identity when generating the name information for any particular database.

The parameters for the generator include the number of links, groups and names, the degree of overlap between the groups and the mean size of the groups and the links. The degree of overlap controls the extent to which entities belong to multiple groups.

Each link 1 in our dataset is generated independently. First, a preference group \( q \) is selected according to the prior probabilities of the groups. Then the number of name for the link is chosen from a normal distribution. Each reference \( r \) is chosen by selecting a name \( a \) from the group \( q \) according their probabilities of belonging to \( q \). Each reference also has a small probability of being selected from any random group \( q' \) different from \( q \). This is the data that we use to evaluate our group detection algorithm.

### 5.7.2 Evaluation Measure

We evaluate our algorithm by measuring the quality of the clusters generated and how accurately we find dis-similarity value. We use two measures of cluster quality dispersion and diversity. For the task of de-duplication, entity dispersion reflects the number of different clusters and
the references corresponding to the same entity name are spread over. A lower
dispersion is better and a perfect de-duplication has dispersion. Cluster
diversity quantifies the number of distinct entities that have been put in the
same cluster. Lower diversity is also preferable as a perfect clustering has
diversity. For the task of group detection using clustering, we similarly
measure group dispersion over clusters and the cluster diversity in terms of
group labels of links included in the same cluster. For both tasks, there is an
inherent trade-off between improving diversity and dispersion; an
improvement in one will usually adversely affect the other. We consider the
weighted average of the dispersion over entities or groups and of the diversity
over clusters as a measure of performance.

5.8 PERFORMANCE ANALYSIS

This algorithm is based on sequential approach. That is, first
generate a cluster and then apply a dis-similarity calculation algorithm to each
cluster value. The proposed algorithms have no disagreement term. The
algorithm produces a good solution for complicated data sets. The proposed
algorithm handled nominal and continuous attribute in natural format and there
is no information loss.

The data sets are formed in a well-known manner and they are used
to compare the performance of the algorithms. The algorithm has multi-stage
decision system with multiple parameters. The proposed algorithm is coded in
java programming language and windows platform with Intel CORE i3 and
1GB RAM. Table 5.1 describes the characteristics of the data set.
<table>
<thead>
<tr>
<th>S. No</th>
<th>Name</th>
<th>Short Name</th>
<th>Account Type</th>
<th>Account No</th>
<th>Region</th>
<th>Cluster No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Venkatesh</td>
<td>V</td>
<td>SB</td>
<td>SB9922</td>
<td>North</td>
<td>1,2</td>
</tr>
<tr>
<td>2</td>
<td>Venkatesh Kumar</td>
<td>VK</td>
<td>SB</td>
<td>SB28109r</td>
<td>South</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Venkatesh Kumar</td>
<td>VK</td>
<td>CR</td>
<td>CR723759</td>
<td>South</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Venketesh Kumar</td>
<td>VK</td>
<td>CD</td>
<td>CD7279</td>
<td>South</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Venkatesh Kumar</td>
<td>VK</td>
<td>DB</td>
<td>DB729875</td>
<td>South</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Venketesh</td>
<td>V</td>
<td>CR</td>
<td>CR124188</td>
<td>North</td>
<td>1,2</td>
</tr>
<tr>
<td>7</td>
<td>Venkatesh Arumugam</td>
<td>VA</td>
<td>SB</td>
<td>SB639386</td>
<td>East</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Venkatesh Arumugam</td>
<td>VA</td>
<td>CR</td>
<td>CR644226</td>
<td>East</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>Venkatesh Arumugam</td>
<td>VA</td>
<td>CD</td>
<td>CD01274</td>
<td>East</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>Venkatesh Arumugam</td>
<td>VA</td>
<td>DB</td>
<td>DB9212221</td>
<td>East</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>Kumar</td>
<td>K</td>
<td>SB</td>
<td>SB65113</td>
<td>West</td>
<td>3,4</td>
</tr>
<tr>
<td>13</td>
<td>Kumar Murugesh</td>
<td>KM</td>
<td>SB</td>
<td>SB124574</td>
<td>South</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>Kumar Murugesh</td>
<td>KM</td>
<td>CR</td>
<td>CR536124</td>
<td>South</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>Kumar Murugash</td>
<td>KM</td>
<td>CD</td>
<td>CD912094</td>
<td>South</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>Kumar Muruge#</td>
<td>KM</td>
<td>DB</td>
<td>DB65187</td>
<td>South</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>Kumar</td>
<td>K</td>
<td>CR</td>
<td>CR7218921</td>
<td>West</td>
<td>3,4</td>
</tr>
<tr>
<td>18</td>
<td>Kumar D</td>
<td>KD</td>
<td>CD</td>
<td>CD7572121</td>
<td>South</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>Kumar D</td>
<td>KD</td>
<td>DB</td>
<td>DB673256</td>
<td>South</td>
<td>4</td>
</tr>
</tbody>
</table>

It shows the sample data with respect to the banking information. It contains information about Name, Short Name, Account Type, Account No, and Region. With respect to name attribute some data contains the typo error as well as missing information with different combination, such as only first token is match, Character missing in the name field, fist token is match and second token is not match etc. With respect to the Account type attribute contains four different types of accounts SB (Saving Bank), CU (Current), CR (Credit), DB (Debit). Region attributes were specified as the following four types South, North, East, and West.
The algorithm is tested on different types of data set for different values of iteration and particle with a range from 5, 10, 15, 20, and 25. The data set contains the missing character and typo error. To find optimal value for a number of particle and iteration we should run the algorithm for various combinations and the predictive accuracy of generated data set using the proposed algorithm for the above data set is tabulated in Table 5.1.

**Table 5.2 Number of iteration and particle required to produce more accurate results for typo error or change of character**

<table>
<thead>
<tr>
<th>Particle/Iteration</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><strong>100</strong></td>
<td>97.41</td>
<td>97.22</td>
<td>96.87</td>
<td>96.5</td>
</tr>
<tr>
<td>10</td>
<td>96.81</td>
<td>97.05</td>
<td>97.18</td>
<td>97.12</td>
<td>97.06</td>
</tr>
<tr>
<td>15</td>
<td>96.58</td>
<td>96.85</td>
<td>97.02</td>
<td>97.15</td>
<td>97.04</td>
</tr>
<tr>
<td>20</td>
<td>96.45</td>
<td>96.79</td>
<td>97.18</td>
<td>97.03</td>
<td>96.72</td>
</tr>
<tr>
<td>25</td>
<td>96.38</td>
<td>96.23</td>
<td>97.06</td>
<td>97.17</td>
<td>96.51</td>
</tr>
</tbody>
</table>

**Table 5.3 Number of iteration and particle required to produce more accurate results for character does not exist**

<table>
<thead>
<tr>
<th>Particle/Iteration</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>80.98</td>
<td>81.75</td>
<td>82.45</td>
<td>82.35</td>
<td>82.29</td>
</tr>
<tr>
<td>10</td>
<td>82.05</td>
<td><strong>85.22</strong></td>
<td>85.01</td>
<td>84.93</td>
<td>84.36</td>
</tr>
<tr>
<td>15</td>
<td>82.15</td>
<td>85.17</td>
<td>85.19</td>
<td>84.89</td>
<td>84.27</td>
</tr>
<tr>
<td>20</td>
<td>82.35</td>
<td>84.39</td>
<td>84.41</td>
<td>84.35</td>
<td>84.21</td>
</tr>
<tr>
<td>25</td>
<td>82.33</td>
<td>84.13</td>
<td>84.09</td>
<td>84.11</td>
<td>84.09</td>
</tr>
</tbody>
</table>

Table 5.2 clearly indicated that the value pair (5,5) for the particle and iteration respectively produce more accurate result for data set contains the information of typo error or change of character. Table 5.3 clearly indicated that the value pair (5,5) for the particle and iteration respectively produced
more accurate result for data set contained the information of missing of character between two string. The above results clearly indicated that the value of number of particles and iteration depended on the characteristics of the data set. It is concluded that the value pair produces accurate results for all types of data set as long as the size of the token is small and it will produce a 100 percent accurate result.

Table 5.4 Total time taken for cluster formation

<table>
<thead>
<tr>
<th>Record Volume</th>
<th>Time(min)</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>10</td>
<td>0.18</td>
</tr>
<tr>
<td>10K</td>
<td>20</td>
<td>0.26</td>
</tr>
<tr>
<td>35K</td>
<td>30</td>
<td>0.39</td>
</tr>
<tr>
<td>50K</td>
<td>40</td>
<td>0.53</td>
</tr>
<tr>
<td>70K</td>
<td>50</td>
<td>1.21</td>
</tr>
<tr>
<td>80K</td>
<td>60</td>
<td>1.43</td>
</tr>
<tr>
<td>100K</td>
<td>70</td>
<td>2.01</td>
</tr>
</tbody>
</table>

Figure 5.3 Total time taken for cluster formation
Table 5.4 and Figure 5.3 show the total time taken for clustering formation of different volumes (5K, 10K, 35K, 50K, 70K, 80K, and 100K) of data. Our executed result clearly shows for clustering formation it takes maximum of around two minutes for 100k of data set volumes.

Table 5.5 Total time taken for Dis-Similarity calculation with Single attribute and Single Token

<table>
<thead>
<tr>
<th>Record Volume</th>
<th>Time(min)</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>10</td>
<td>0.31</td>
</tr>
<tr>
<td>10K</td>
<td>20</td>
<td>1.02</td>
</tr>
<tr>
<td>35K</td>
<td>30</td>
<td>1.46</td>
</tr>
<tr>
<td>50K</td>
<td>40</td>
<td>2.59</td>
</tr>
<tr>
<td>70K</td>
<td>50</td>
<td>5.07</td>
</tr>
<tr>
<td>80K</td>
<td>60</td>
<td>6.27</td>
</tr>
<tr>
<td>100K</td>
<td>70</td>
<td>9.03</td>
</tr>
</tbody>
</table>

Figure 5.4 Total times taken for Dis-Similarity calculation with Single attribute and Single Token
Table 5.5 and Figure 5.4 show Total time taken for Dis-Similarity calculation with Single attribute and Single Token of different volumes (5K, 10K, 35K, 50K, 70K, 80K, and 100K) of data. Our executed result clearly shows for calculating dis-similarity value it takes maximum of around ten minutes for 100k of data set volumes.

**Table 5.6 Total time taken for Dis-Similarity calculation with Single attribute and Multiple Token**

<table>
<thead>
<tr>
<th>Record Volume</th>
<th>Time(min)</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>10</td>
<td>0.43</td>
</tr>
<tr>
<td>10K</td>
<td>20</td>
<td>1.56</td>
</tr>
<tr>
<td>35K</td>
<td>30</td>
<td>3.01</td>
</tr>
<tr>
<td>50K</td>
<td>40</td>
<td>4.09</td>
</tr>
<tr>
<td>70K</td>
<td>50</td>
<td>6.53</td>
</tr>
<tr>
<td>80K</td>
<td>60</td>
<td>9.08</td>
</tr>
<tr>
<td>100K</td>
<td>70</td>
<td>12.47</td>
</tr>
</tbody>
</table>

**Figure 5.5** Total time taken for Dis-Similarity calculation with Single attribute and Multiple Token
Table 5.6 and Figure 5.5 show Total time taken for Dis-Similarity calculation with Single attribute and Multiple Token of different volumes (5K, 10K, 35K, 50K, 70K, 80K, and 100K) of data. Our executed result clearly shows for calculating dis-similarity value it takes maximum of around ten minutes for 100k of data set volumes.

Table 5.7 Total Turnaround Time for Cluster Formation and Dis-Similarity calculation with Single attribute and Single Token

<table>
<thead>
<tr>
<th>Record Volume</th>
<th>Time(min)</th>
<th>Total-Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>10</td>
<td>0.49</td>
</tr>
<tr>
<td>10K</td>
<td>20</td>
<td>1.28</td>
</tr>
<tr>
<td>35K</td>
<td>30</td>
<td>2.25</td>
</tr>
<tr>
<td>50K</td>
<td>40</td>
<td>3.52</td>
</tr>
<tr>
<td>70K</td>
<td>50</td>
<td>6.28</td>
</tr>
<tr>
<td>80K</td>
<td>60</td>
<td>8.1</td>
</tr>
<tr>
<td>100K</td>
<td>70</td>
<td>11.04</td>
</tr>
</tbody>
</table>

Figure 5.6 Total Turnaround Time for Cluster Formation and Dis-Similarity calculation with Single attribute and Single Token
Table 5.7 and Figure 5.6 show Total Turnaround time taken for Cluster Formation and Dis-Similarity calculation with Single attribute and Multiple Token of different volumes (5K, 10K, 35K, 50K, 70K, 80K, and 100K) of data. Our executed result clearly shows for calculating dis-similarity value it takes maximum of around Ten minutes for 100k of data set volumes. Our previous chapter executed result of 100K volumes takes around 25 minute. When compared with the previous result here we have proved admirable performance. We have also reduced more than 150% of total turnaround time.

Table 5.8 Total Turnaround Time for Cluster Formation and Dis-Similarity calculation with Single attribute and Multiple Token

<table>
<thead>
<tr>
<th>Record Volume</th>
<th>Time(min)</th>
<th>Total-Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>5K</td>
<td>10</td>
<td>1.01</td>
</tr>
<tr>
<td>10K</td>
<td>20</td>
<td>2.24</td>
</tr>
<tr>
<td>35K</td>
<td>30</td>
<td>3.4</td>
</tr>
<tr>
<td>50K</td>
<td>40</td>
<td>5.02</td>
</tr>
<tr>
<td>70K</td>
<td>50</td>
<td>8.14</td>
</tr>
<tr>
<td>80K</td>
<td>60</td>
<td>10.51</td>
</tr>
<tr>
<td>100K</td>
<td>70</td>
<td>14.48</td>
</tr>
</tbody>
</table>

Figure 5.7 Total Turnaround Time for Cluster Formation and Dis-Similarity calculation with Single attribute and Multiple Token
Table 5.8 and Figure 5.7 show Total Turnaround times taken for Cluster Formation and Dis-Similarity calculation with Single attribute and Multiple Token of different volumes (5K, 10K, 35K, 50K, 70K, 80K, and 100K) of data. Our executed result clearly shows that for calculating dis-similarity value it takes maximum of around 15 minutes for 100k of data set volumes. Our previous chapter executed result of 100K volumes takes around one hour. When compared with the previous result here we have proved admirable performance. We have also reduced more than 300% of total turnaround time.

5.9 SUMMARY

In this chapter, proposed grouping induction algorithm for dis-similarity calculation has been analyzed. For group simulation we have implemented Alternative Decision Tree technique and first level validate the extraction data, second level based on short name form the grouping. The simulation results show better results than the previous chapter executed result. As the various combination values were defined to the parameter, several interesting phenomena have been observed. From the observations, it is proved that the total time taken and number of iteration are fully depending on the characteristics of the data set. The performance of this algorithm duplicate detection with grouping has been compared in the next chapter both experimentally as well as by means of statistical test. Consequently, it becomes clear that the duplicate detection with grouping algorithm performs well.