Chapter 4: Text Clustering

4.1 Introduction to Text Clustering

Clustering is an unsupervised method of grouping texts / documents in such a way that in spite of having little knowledge about the content of the documents, we can group together similar documents into independent clusters based on some input parameters. In fact, given a training data set of documents, the goal of a clustering algorithm is to group similar documents in the same cluster while putting dissimilar documents in different clusters. Clustering is used in a wide variety of fields: biology, statistics, pattern recognition, information retrieval, machine learning, psychology, and Data Mining. For example, it is used to group related documents for browsing, to find genes and proteins that have similar functionality, to find the similarity in medical image database, or as a means of data compression. Document clustering has been studied for quite a while and has wide applications like topic extraction, content filtering and also as a pre-processing step for text categorization. The indexing methods that I have used to rank documents have already been discussed in the previous sections. The measures that have been used in implementing the clustering algorithms are the tf-idf scores, singular value decomposition (svd), etc. and distance measure like the cosine similarity, Jaccard’s co-efficient, etc.

4.2 Evaluation of Cluster Quality

There are different measures available to evaluate the correctness of clusters after an algorithm has been implemented. There are two types of measures – internal and external. Internal measures check the correctness within the clusters and across clusters i.e. how similar documents within a single cluster are and how different documents are across clusters. The other more popular measures are the external measures. They are called external measures because we test the clusters on documents which have
been already classified (training sets) or those classified by human experts/judges which are called the gold standard classes.

There are many other external measures. Some are explained as follows:\(^1\):

- **Purity**: This is a very simple method. To compute purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned documents and dividing by \( N \) – total number of documents.

\[
purity(\Omega, C) = \frac{1}{N} \sum_k \max_j | w_k \cap c_j |
\]  

(4.1)

Where,

\( \Omega = \{w_1, w_2, ..., w_k\} \) is the set of clusters

\( C = \{c_1, c_2, ..., c_j\} \) is the set of classes

\( w_k \) - the set of documents in \( w_k \)

\( c_j \) - the set of documents in \( c_j \)

Purity is close to zero for bad clusters and close to one for good clusters.

- **The Rand Index (RI)**: The Rand index penalizes both false positive and false negative decisions during clustering. A true positive (TP) decision assigns two similar documents to the same cluster; a true negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors we can commit. A false positive (FP) decision assigns two dissimilar documents to the same cluster. A false negative (FN) decision assigns two similar documents to different clusters. The Rand index measures the percentage of decisions that are correct. That is, it is simply accuracy.

\[
RI = \frac{TP + TN}{TP + FP + FN + TN}
\]  

(4.2)

---

\(^1\) These measures are taken from the book “An Introduction to Information Retrieval” by Christopher et al, online edition
F-measure: This measure is based on recall and precision. The formula to calculate the F-measure for a cluster \( j \) and class \( i \) is:

\[
\text{Recall}(i, j) = \frac{n_{ij}}{n_i}
\]

(4.3)

\[
\text{Precision} (i, j) = \frac{n_{ij}}{n_j}
\]

(4.4)

Where,

\( n_{ij} \) – number of members of class \( i \) in cluster \( j \)

\( n_i \) – number of members of class \( i \)

\( n_j \) – number of members of class \( j \)

\[
F(i, j) = \frac{2 \times \text{Recall}(i, j) \times \text{Precision} (i, j)}{\text{Recall}(i, j) + \text{Precision} (i, j)}
\]

(4.5)

After finding the F measure for all clusters in this way the overall F measure is computed by taking the weighted average of all values for F:

\[
F = \sum_i \frac{n_i}{n} \left\{ F(i, j) \right\}
\]

(4.6)

Where,

\( n \) – total number of documents

Another way of representing the above is:

\[
\text{Precision} (P) = \frac{TP}{TP + FP}
\]

(4.7)

\[
\text{Recall} (R) = \frac{TP}{TP + FN}
\]

(4.8)

\[
F = \frac{2PR}{P+R}
\]

(4.9)

Other than these measures there are others like the Mutual Information (MI) and the Entropy – both of which are based on probabilities.

The clustering algorithms studied are:

- The K-Means Algorithm
- The DBSCAN Algorithm
- The SNN
- Have developed SNNAE
4.3 The K-Means Algorithm

4.3.1 The simple K-Means

The K-Means is a partitioning method of clustering where n documents are partitioned into k partitions / clusters in such a way that each cluster has at least one document and each document belongs to only one cluster. The second condition is sometimes relaxed if we know that a document can belong to more than one topic or subject.

This is one of the simplest methods and creates clusters which are spherical in shape. This algorithm works well for a small corpus. The time complexity of this algorithm is linear in the number of documents. K-means is based on the concept that a center point can represent a cluster. In particular, for K-means we use the notion of a centroid, which is the mean or median point of a group of points (in this case documents). Note that a centroid almost never corresponds to an actual data point.

In the K-Means algorithm the input is the number of clusters k, the corpus containing the documents to be clustered and k initial arbitrary documents. The output contains k clusters of documents.

The algorithm works as follows:

1. Select arbitrarily K documents as the initial centroids.
2. Assign each document to the closest centroid using some similarity function.
3. Re-compute the centroid (mean) of each cluster.
4. Repeat steps 2 and 3 until the centroids do not change.

4.3.2 The Bisecting K-Means Algorithm

This algorithm is combination of K-Means and agglomerative hierarchical algorithm (uses the divisive method). This algorithm also has a complexity which is linear in the number of documents.

In this method we start with a single cluster which contains all the documents. The algorithm then splits the main cluster in different clusters as per the following algorithm:

1. Select a cluster to split
2. Find two sub-clusters from the cluster using the K-Means algorithm
3. Repeat step 2, the bisecting step, by selecting the largest cluster with least overall similarity.

4. Repeat steps 1, 2 and three until the desired number of clusters is reached.

4.3.3 The similarity measures

To find the distances between the documents and the centroids, I first calculated the tf-idf scores for each term in the documents. After that I created a term X document matrix for the tf-idf scores.

As per the details given in Table 2, for different values of \( k \) and initial cluster centroids, the clusters created are shown. The similarity measure used to find the similarity between the documents and the centroids was the cosine similarity.

\[
(doci, docj) = \frac{\sum_{1}^{k} (doci^{(k)} \times docj^{(k)})}{\|doci\| \times \|docj\|}
\]  
(4.10)

Where,

\[
\|doci\| = \sqrt{\sum_{1}^{k} (doci^{(k)})^2}
\]

(used to normalize the vectors)

doci \& docj – are the tf-idf scores of the two documents

For our algorithm, we need to find the centroid \( c \) once iteration is complete. The centroid for a set of \( S \) documents with their vector representations can be found by,

\[
c = \frac{1}{|S|} \sum_{d \epsilon S} d
\]

(4.11)

This would be a vector obtained by averaging individual scores of all documents belonging to the set \( S \). Finally the sets become the clusters.
4.4 The DBSCAN Algorithm

It stands for Density Based Spatial Clustering Algorithm with Noise. Though this is a density based algorithm it has been found to give very good results in text clustering also.

DBSCAN requires two parameters: epsilon (eps) and minimum points (minPts). In the textual data eps would be the value of cosine distance (between 0 and 1) and minPts generally works well for 4 (at least four documents are similar to the document under consideration). For the following algorithm, a point is a document.

1. It starts with an arbitrary starting point that has not been visited. It then finds all the neighbor points within distance eps of the starting point.
2. If the number of neighbors is greater than or equal to minPts, a cluster is formed. The starting point and its neighbors are added to this cluster and the starting point is marked as visited.
3. The algorithm then repeats the evaluation process for all the neighbors recursively.
4. If the number of neighbors is less than minPts, the point is marked as noise.
5. If a cluster is fully expanded (all points within reach are visited) then the algorithm proceeds to iterate through the remaining unvisited points in the dataset.

This algorithm has an advantage that it does not require to know the number of clusters in the data a priori and it can also detect noise. Moreover the clusters unlike K-Means are of arbitrary shape.

The disadvantage is it depends on the distance function. For structured low dimensional data the Euclidean distance is good enough but for the textual data we sometimes consider other similarity measures also. I have used the cosine similarity to implement DBSCAN. If documents of a particular class are few in number they would get classified as noisy documents.

4.5 The Shared Nearest Neighbor Algorithm (SNN)

The main difference between this algorithm and DBSCAN is that it defines the similarity between points by looking at the number of nearest neighbors that
two points share. Using this similarity measure in the SNN algorithm, the density is defined as the sum of the similarities of the nearest neighbors of a point. Points with high density become core points, while points with low density represent noise points. All remainder points that are strongly similar to a specific core points will represent a new clusters.

The steps to implement SNN are:

1. Identify the k nearest neighbors for each point (the k points most similar to a given point, using a distance function to calculate the similarity).
2. Calculate the SNN similarity between pairs of points as the number of nearest neighbors that the two points share. The SNN similarity is zero if the second point in not in its list of k nearest neighbors, and vice-versa.
3. Calculate the SNN density of each point: number of nearest neighbors that share Eps or greater neighbors.
4. Detect the core points. If the SNN density of the point is equal or greater than MinPts then classify the point as core.
5. Form the cluster from the core points. Classify core points into the same cluster if they share Eps or greater neighbours.
6. Identify the noise points. All non-core points that are not within a radius of Eps of a core point are classified as noise.
7. Assign the remainder points to the cluster that contains the most similar core point.

The SNN similarity takes the sum of the similarity of the point's nearest neighbors as a measure of density. It works well for the data in low, medium and high dimensionality. The SNN similarity measure reflects the local configuration of the points in the data space. It is insensitive to the variation of density and dimensionality. According to SNN density, the higher the density, it is likely to represent core or representative point and lower the density, it is likely to represent noise points. The key feature of SNN density measure is that it is able to find clusters of different shapes and sizes.
After studying the K-Means, DBSCAN and SNN algorithms I have designed a new algorithm which I call the SNNAE (Shared Nearest Neighbor Algorithm with Enclosures). This proposed algorithm has created better clusters and given a better output.

This algorithm has been published by the World Research Congress and IEEE and is available on IEEE Computer Society Portal, ACM Digital Library, Google Scholar, Bibsonomy, etc. and is available on the site: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5171034

4.6 The Shared Nearest Neighbor Algorithm with Enclosures (SNNAE)

The proposed algorithm, SNNAE, is based on the ‘enclosure’ approach, which uses an inexpensive distance measure to approximately divide the data into overlapping subsets and then applies expensive distance measure to calculate similarity only between the points in the same enclosure. The proposed algorithm is efficient and scalable because with this approach significant computation is reduced by eliminating all of the distance comparisons among points that do not fall within a common enclosure.

In the proposed algorithm, the density of the points is defined in terms of number of neighbors with which it shares a total number – Eps(a parameter) or greater neighbors. If this number is greater or equal to the MinPts(another parameter), then a point is considered to have high density otherwise it represents low density points. Also, the parameter Eps is calculated automatically from the enclosures and MinPts can be user specified or fixed to 3, which is shown as a good value from many experimental results. The size of nearest neighborhood is also provided as input. The steps of the algorithm are as follows:

Creating Enclosures: In the first stage, data points are divided into overlapping subset or enclosures where enclosure is simply a subset of the data points. Every data point must appear in at least one enclosure and any data point may appear in more than one enclosure as shown in
Figure 4.1\textsuperscript{2}. Enclosures are created with the intention that points not appearing in any common enclosure are far enough apart that they could not possibly be in the same cluster.

![Diagram of data clusters and enclosures]

**Figure 4.1 Three data clusters and enclosures**

In the above figure, the solid circles show the example of overlapping enclosures that cover all the data points. Consider an arbitrary point p as shown in the figure. All the points inside the solid circle are the nearest adjacent points for the central point p. All the points between the dashed circle and solid circle are the nearest far adjacent points to the central point p. The dashed circle is used to ensure that points in the same clusters will not be split into different clusters. For e.g. there are no common points in two overlapping enclosures of cluster1. If dashed circles were not used then natural cluster1 would split into two small clusters.

In the second stage, the proposed algorithm finds the optimal value of Eps, the radius of neighborhood to define the density from the overlapped enclosures created in first stage and then apply the SNN clustering algorithm.

To create overlapping subset or enclosure we first need to calculate radius of the enclosure. First of all, all the data points are stored in a single cluster called cluster feature (CF). This is a data structure which contains summary information about all points.

\[ \overline{CF} = \left( n, \overline{LS} \right) \]  
\[ (4.12) \]

Where, \( LS \) is the linear sum of the \( n \) data points.

\[ \overline{LS} = \sum_{i=1}^{n} \overline{x_i} \]  
\[ (4.13) \]

Where, \( x_i \) is \( d \)-dimensional data points.

Then to calculate the radius of the data space which covers all the data point, we first find the centre of all the data point using the formula:

\[ \overline{x_0} = \frac{\overline{LS}}{n} = \frac{\sum_{i=1}^{n} \overline{x_i}}{n} \]  
\[ (4.14) \]

Then radius of entire data set is calculated as:

\[ R = \left( \sum_{i=1}^{n} (\overline{x_i} - \overline{x_0})^2 / n \right)^{1/2} \]  
\[ (4.15) \]

From this radius, area of the circle is found as:

**circular area = 3.14 \times R^d**  
\[ (4.16) \]

The circular area for more than two dimensions requires the \( 4/3 \) coefficient to be included in the formula. Then we calculate area from another point of view, called rectangular area, based on minimum bounding rectangle, which also covers all data point.

\[ \text{rectangular area} = \prod_{i=1}^{d} L_i \]  
\[ (4.17) \]

Where, \( L_i \) is the difference between maximum and minimum value for the dimension \( i \), which is also called as length of dimension \( i \). In the Figure 4.2 blue points indicate length of \( x \) dimension and red points indicate length of \( y \) dimension.

In the multi-dimensional data set, as the dimension increase the data becomes more uniform and sparser. Therefore, while calculating radius of the overlapping enclosures, we have to consider dimension of the
data set. Also, radius depends on the area of the data space. This ratio should vary between 0 and 1.

\[
\text{circular area} = \text{area of the circle}
\]

\[
\text{rectangular area} = \text{area of the rectangle}
\]

**Figure 4.2 Circular and rectangular area of data space**

By considering ratio area and dimension of the data space, the radius \( r \) of the overlapped enclosures is calculated as:

\[
r = d \times \text{ratio area} + \frac{\text{ratio area}}{2}
\]  \hspace{1cm} (4.18)

Where, \( d \) is the dimension of data space and ratio area is ratio of rectangular and circular area or inverse. Always, the radius of the overlapped enclosures is greater than expected Eps.

Overlapping enclosures are created using the radius \( r \). To create enclosures, the first point of the data set is taken as the centre of the first enclosure. Then consider distance between every other point with that centre point. All the points whose distance to the centre point is less than or equal to radius \( r \), are considered to be the nearest adjacent point to the centre point and are put in the list1 of that enclosure. If the distance to the centre point is less than or equal to \( r \times 1.5 \) then points are considered to be as nearest far adjacent points and are put in list2 of that enclosure. All the points whose distance is greater than \( r \times 1.5 \) and less than \( r \times 2 \) are considered to be as centre of the next enclosure to ensure overlapping of enclosures.

For each point in the list1, the algorithm keeps the distance to the nearest enclosure and identification of that enclosure, since point may be covered by more than one enclosure.
Finding Optimal Value of Eps: To find the optimal value of Eps, we consider only the points in the solid threshold. We find distance for each point with every other point within the same enclosure. These distances are calculated by using the following equation:

\[ d(P_i, P_j) = \left( \sum_{k=1}^{d} (P_{i,k} - P_{j,k})^2 \right)^{1/2} \]  

(4.19)

Where, \( d \) is the dimension, and \( P_{i,k} \) and \( P_{j,k} \) are the \( k \)th component of the \( i \)th and \( j \)th object. From this distance, we find the maximum distance between the nearest pair. This process is repeated for all the enclosures. Then optimal value of Eps is found by taking average of all of these maximum distances, i.e. \( Eps = \frac{\text{max}}{k} \), where \( k \) is the number of overlapped enclosures and \( \text{max} \) is the maximum distance for enclosure \( i \). This Eps value is used to measure the density of the point in the proposed algorithm.

Finding Nearest Neighbor: When the algorithm retrieves the nearest neighbors of a point, it directly goes to the best enclosure (i.e. nearest enclosure which covers that point.). It then computes the distance with every other point in enclosure (within solid threshold) and considers only those points that are having distance less than or equal to Eps as the nearest neighbors of that point. If size \( k \), of the nearest neighbor list is given as an input then only \( k \) nearest neighbors are considered, otherwise all neighbors having distance less than or equal to Eps are considered as nearest neighbors of that point.

If the distance between the centre point and point considered is greater than Eps then the algorithm computes the distance between that point and solid edge. If this distance is less than Eps value then we compute distance between the point considered and all the points in the dashed edge to find its nearest neighbors with respect to Eps. In this way the final clusters are created.
Chapter 4: Text Clustering

4.6.1 Complexity of the SNNAE Algorithm

The complexity of the algorithm can be found as follows:

Let,
\[ n = \text{number of records/instances/data points in data file} \]
\[ d = \text{number of dimensions/attributes} \]
\[ k = \text{size of nearest neighbor list} \]
\[ e = \text{number of enclosures} \]
\[ s = \frac{n}{e} \text{i.e. average number of points in each enclosure} \]

The proposed algorithm is divided into two stages.

1. In the first stage, \( n \) data points are divided into \( e \) enclosures. So its Complexity is \( O(ne) \)

2. The Eps value is calculated using only the points that are amongst the same enclosure. Assuming each enclosure covers \( s \) points on an average, calculation of Eps will have time complexity of \( (s^2e) \). So its Complexity is \( O(s^2e) \).

So, the time complexity of the first stage is:

\[ O(ne + s^2e) \]

In the second stage, getting the nearest neighbor, complexity:

\[ O(ns) \]

Since distance of each point with every other point in the same enclosure is only calculated. Therefore, the total time complexity of the proposed algorithm is:

\[ O(ne + s^2e + ns) \]

4.6.2 The Dataset Description

The datasets used those already mentioned in the first chapter of Text Mining Overview. In the implementation of the algorithms, the datasets are db1, fish, abalone, and cpu. These datasets vary in size and number of attributes (dimensions). The site from where they can be downloaded is http://archive.ics.uci.edu/ml/ and the WEKA datasets. The brief description of each dataset used in the evaluation of this algorithm and the SNN algorithm is shown in Table 4.1.
### Table 4.1 Details of datasets used

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instances</th>
<th>Attributes</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpu</td>
<td>209</td>
<td>7</td>
<td>Real</td>
</tr>
<tr>
<td>Fish</td>
<td>1100</td>
<td>2</td>
<td>Synthetic</td>
</tr>
<tr>
<td>Abalone</td>
<td>4177</td>
<td>7</td>
<td>Real</td>
</tr>
<tr>
<td>Db1</td>
<td>10000</td>
<td>2</td>
<td>Synthetic</td>
</tr>
</tbody>
</table>

#### 4.6.3 Implementation and result

The proposed SNNAE algorithm has been evaluated on several different real and synthetic datasets. The result of this algorithm is compared with that of SNN clustering algorithm in terms of scalability, efficiency and quality of clusters. Both algorithms produce the same result most of the time, but compared to SNN, SNNAE is more scalable and efficient.

The implementation shown is on the Abalone dataset. Abalones are small to very large-sized edible sea snails. The shells of abalones have a low and open spiral structure, and are characterized by several open respiratory pores in a row near the shell's outer edge. The flesh of abalones is widely considered to be a desirable food, and is consumed raw or cooked in a variety of different dishes. The attribute details are given in Table 4.2. The algorithms were implemented by varying the values of the input parameters and finding the output for each case. Some sample inputs and outputs are given further.

The parameter values calculated using the SNNAE algorithm for creating first the enclosures and then finding the Eps is depicted in Table 4.3. In Table 4.4 a comparison in terms of execution time in seconds between the SNN and the SNNAE algorithm at each step of execution is shown.

There are seven attributes and a total of 4177 record sets. The details of the attributes are:
Table 4.2 Attribute details of Abalone dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>continuous</td>
<td>Longest shell measurement</td>
</tr>
<tr>
<td>Diameter</td>
<td>continuous</td>
<td>perpendicular to length</td>
</tr>
<tr>
<td>Height</td>
<td>continuous</td>
<td>with meat in shell</td>
</tr>
<tr>
<td>Whole weight</td>
<td>continuous</td>
<td>whole abalone</td>
</tr>
<tr>
<td>Shucked weight</td>
<td>continuous</td>
<td>weight of meat</td>
</tr>
<tr>
<td>Viscera weight</td>
<td>continuous</td>
<td>gut weight (after bleeding)</td>
</tr>
<tr>
<td>Shell weight</td>
<td>continuous</td>
<td>after being dried</td>
</tr>
</tbody>
</table>

Input: Dataset, Minimum points (Minpts), nearest neighbors (nnls)
Output: Clusters, noise points

Table 4.3 Value of different parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of all data</td>
<td>0.489</td>
</tr>
<tr>
<td>Radius of enclosure</td>
<td>0.099</td>
</tr>
<tr>
<td>Number of enclosures</td>
<td>288</td>
</tr>
<tr>
<td>Eps</td>
<td>0.0538</td>
</tr>
</tbody>
</table>

Case No. 1: Considered Minpts = 3, and all possible nearest neighbours.

Table 4.4 Implementation Result for minpts-3, nnls-all

<table>
<thead>
<tr>
<th>No.</th>
<th>Functionality</th>
<th>SNN time (sec)</th>
<th>SNNAE time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Find nearest neighbors</td>
<td>51.262</td>
<td>4.227</td>
</tr>
<tr>
<td>2</td>
<td>Find shared nearest neighbors</td>
<td>92.305</td>
<td>76.675</td>
</tr>
<tr>
<td>3</td>
<td>Get initial clusters</td>
<td>3.011</td>
<td>2.917</td>
</tr>
<tr>
<td>4</td>
<td>Get border and noise points</td>
<td>68.531</td>
<td>75.973</td>
</tr>
<tr>
<td>5</td>
<td>Compose cluster</td>
<td>0.375</td>
<td>0.764</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>215.484</td>
<td>160.556</td>
</tr>
</tbody>
</table>
Chapter 4: Text Clustering

Figure 4.3 Implementation graph for minpts-3 and nnls-all

Table 4.5 Points distribution in clusters with minpts-3, nnls-all

<table>
<thead>
<tr>
<th>Cluster No.</th>
<th>% of points with SNN</th>
<th>% of points with SNNAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (noise)</td>
<td>14.29</td>
<td>15.21</td>
</tr>
<tr>
<td>1</td>
<td>85.08</td>
<td>84.17</td>
</tr>
<tr>
<td>2</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>4</td>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Chapter 4: Text Clustering

**Case No. 2:** Considered Minpts = 3, and nnls = 20

Table 4.6 Implementation Results for minpts-3, nnls-20

<table>
<thead>
<tr>
<th>No.</th>
<th>Functionality</th>
<th>SNN time (sec)</th>
<th>SNNAE time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Find nearest neighbors</td>
<td>49.561</td>
<td>4.259</td>
</tr>
<tr>
<td>2</td>
<td>Find shared nearest neighbors</td>
<td>5.257</td>
<td>0.842</td>
</tr>
<tr>
<td>3</td>
<td>Get initial clusters</td>
<td>1.139</td>
<td>1.077</td>
</tr>
<tr>
<td>4</td>
<td>Get border and noise points</td>
<td>69.514</td>
<td>74.943</td>
</tr>
<tr>
<td>5</td>
<td>Compose cluster</td>
<td>0.468</td>
<td>0.811</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>125.939</td>
<td>81.932</td>
</tr>
</tbody>
</table>

Figure 4.4 Implementation graph for minpts-3 and nnls-20.
Table 4.7 Points distribution in clusters with minpts-3, nnls-20

<table>
<thead>
<tr>
<th>Cluster No.</th>
<th>% of points with SNN</th>
<th>% of points with SNNAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (noise)</td>
<td>14.01</td>
<td>15.87</td>
</tr>
<tr>
<td>1</td>
<td>84.68</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td>0.34</td>
<td>0.22</td>
</tr>
<tr>
<td>3</td>
<td>0.22</td>
<td>0.12</td>
</tr>
<tr>
<td>4</td>
<td>0.12</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>0.12</td>
</tr>
<tr>
<td>6</td>
<td>0.12</td>
<td>0.07</td>
</tr>
<tr>
<td>7</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>0.1</td>
<td>0.14</td>
</tr>
<tr>
<td>9</td>
<td>0.14</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>0.1</td>
<td>0.17</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Case No. 3: Considered Minpts = 3, and nnls = 25

Table 4.8 Implementation Results for minpts-3, nnls-25

<table>
<thead>
<tr>
<th>No.</th>
<th>Functionality</th>
<th>SNN time (sec)</th>
<th>SNNAE time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Find nearest neighbors</td>
<td>49.546</td>
<td>4.29</td>
</tr>
<tr>
<td>2</td>
<td>Find shared nearest neighbors</td>
<td>7.129</td>
<td>1.326</td>
</tr>
<tr>
<td>3</td>
<td>Get initial clusters</td>
<td>1.498</td>
<td>1.404</td>
</tr>
<tr>
<td>4</td>
<td>Get border and noise points</td>
<td>65.957</td>
<td>69.031</td>
</tr>
<tr>
<td>5</td>
<td>Compose cluster</td>
<td>0.437</td>
<td>0.796</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>124.582</td>
<td>76.847</td>
</tr>
</tbody>
</table>
In this way two more cases were taken with minpts = 4 and nnls = 20 and then 25. The consolidated result is as shown in Table 4.10.
Table 4.10 Combined Results of all cases

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Minpts</th>
<th>nnls</th>
<th>SNN Total time(sec)</th>
<th>SNNAE Total time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>all</td>
<td>215.484</td>
<td>160.556</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>20</td>
<td>125.939</td>
<td>81.932</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>25</td>
<td>124.582</td>
<td>76.847</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20</td>
<td>127.016</td>
<td>80.793</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>25</td>
<td>128.81</td>
<td>79.076</td>
</tr>
</tbody>
</table>

Similar results have been obtained for the rest of the datasets with structured as well as unstructured data. It is clearly evident that the time taken in finding the nearest neighbors and shared nearest neighbors is very less in SNNAE as compared to SNN. In finding the initial clusters and composing the clusters both take almost the same time. In finding the border and noise points however, SNN is slightly better than SNNAE. The overall time taken is by SNNAE is quite less as compared to SNN (Figure 4.6).
4.7 Conclusion and comparison of algorithms

The proposed algorithm, SNNAE is based on density, and k-nearest neighbor approach. The basic idea of this algorithm is to find the way of computing the nearest neighbors of points by restricting the number of points considered. This algorithm uses the enclosure approach to divide the data into overlapping region which greatly reduce the number of distance calculations required for clustering. This reduces the computational complexity of the SNN clustering algorithm which is $O(n^2)$ to $O(ne + s2e + ns)$. The experimental results demonstrated the scalability and efficiency of the proposed algorithm.

The algorithm has been tested against structured as well as unstructured data. The datasets can be downloaded from the sited mentioned in chapter 1. The proposed algorithm provides a robust alternative to the other considered clustering approaches that are more limited in the types of data and clusters that they can handle.

Table 4.11 shows the comparison of the clustering algorithms like the K-Means, DBSCAN, SNN and SNNAE. The comparison is in terms of complexity, handling multidimensional data etc.

There are many clustering algorithms but the partitioning and hierarchical methods are more popular. Variants of the basic k-Means are also very popular. This chapter gives details of the popular algorithms being used in the field of Text Mining.
Table 4.11 Comparison of clustering algorithms

<table>
<thead>
<tr>
<th>Clustering Criteria</th>
<th>K-means</th>
<th>DBSCAN</th>
<th>SNN</th>
<th>SNNAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>O(nkt) where n = no. of data points k = no. of clusters t = no. of iterations</td>
<td>O(n2) where n is the number of data points</td>
<td>O(n2) where n is the number of data points</td>
<td>O (ne + s2e + ns) where n=no. of data points s= avg. number of points in enclosure e= number of enclosures</td>
</tr>
<tr>
<td>Handle multidimensional data</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Handle large dataset</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Handle Noise</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Shape of Clusters</td>
<td>Spherical only</td>
<td>Any arbitrary shape</td>
<td>Any arbitrary shape</td>
<td>Any arbitrary shape</td>
</tr>
<tr>
<td>Types of data handled</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>Scalable</td>
<td>Yes</td>
<td>Not without enhancement because of its complexity</td>
<td>Not without enhancement because of its complexity</td>
<td>Yes</td>
</tr>
<tr>
<td>Input parameters</td>
<td>k – no. of clusters k initial cluster centroid</td>
<td>Eps – radius Minpts – no. of minimum points</td>
<td>k - size of nearest neighbor list MinPts Eps</td>
<td>Minpts – no. of minimum points Nnls – size of nearest neighbor list</td>
</tr>
</tbody>
</table>

4.8 Future Enhancement

There is a lot of scope in this field of Text Clustering. Methods like the fuzzy clustering are becoming popular as these methods apply the concept of fuzziness i.e. a three valued logic like true, false and maybe for a document to belong to a cluster. There are other hierarchical methods also where still there is scope for research. There are clustering methods related to neural networks also.

Document clustering is still not a very popular method in Information Retrieval. The reason being clustering is slow for large corpora. This can act as a domain of research too.