Effective Term Based Text Clustering Algorithms

P. Ponmuthuramalingam
Department of Computer Science,
Government Arts College, Coimbatore, India

T. Devi
School of Computer Science and Engineering
Bharathiar University, Coimbatore, India

Abstract

Text clustering methods can be used to group large sets of text documents. Most of the text clustering methods do not address the problems of text clustering such as very high dimensionality of the data and understandability of the clustering descriptions. In this paper, a frequent term based approach of clustering has been introduced; it provides a natural way of reducing a large dimensionality of the document vector space. This approach is based on clustering the low dimensionality frequent term sets and not on clustering high dimensionality vector space. Four algorithms for effective term based text clustering has been presented. An experimental evaluation on classical text documents as well as on web documents demonstrates that the proposed algorithms obtain clustering of comparable quality significantly more efficient than existing text clustering algorithms.

Keywords: Frequent term sets, Document clustering, Text documents, Document mining, Text mining, Text clustering.

I. INTRODUCTION

In every industry, almost all the documents on paper have their electronic copies. This is because, the electronic format provides safer storage and occupies much smaller space [6][12][10]. Also, the electronic files provide a quick access to these documents. The text database which consists of documents is usually very large and it becomes a huge challenge to understand hidden patterns or relations in the data [6][9]. As text data is not in numerical format, it cannot be analyzed with statistical methods. However, everyday, people encounter a large amount of information and store or represent it as data, for further analysis and management.

Document clustering has been investigated in different areas of text mining and information retrieval. Document clustering has been studied intensively because of its wide application in areas such as Web Mining [17], Search Engine and Information Retrieval [6][12]. Document clustering is the automatic organization of documents into clusters or groups, so that, documents within a cluster have high similarity in comparison to one another, but are very dissimilar to documents in other clusters [9]. In other words, the grouping is based on the principle of maximizing intra-cluster similarity and minimizing inter-cluster similarity [2][8][16]. The major challenge of clustering is to efficiently identify meaningful groups that are concisely annotated [5][14][16].

In order to increase the precision of the retrieval result, many methods have been proposed [3][15]. By clustering the text documents, the documents sharing the same topic are grouped together. When the clusters are returned, the user can select the group that interests the user most. This method makes the search engine more efficient and accurate.

II. REVIEW OF LITERATURE

Fung B.C.M. (2003) addressed the problem of poor clustering accuracy due to the incorrect estimation of the number of clusters. An algorithm, namely, Frequent Item Set based Hierarchical Clustering (FIHC) proposed by Fung B.C.M. makes use of frequent item set and construction of a hierarchical topic tree from the clusters. A frequent item set is being used as preliminary step and the dimension of each document is drastically reduced, which in turn increases efficiency and scalability [4][7][16]. The author performed the experiment on a Pentium III 667 MHz PC with largest datasets (Reuters) and the proposed algorithm is more scalable because the experiment with 10000 documents shows that FIHC algorithm completes its whole process within two minutes while Unweighted Pair Group Method with Arithmetic Mean (UPGMA) and Hierarchical Frequent Term based Clustering (HFTC) could not even produce a clustering solution [4][15].

Bi-secting k-means generate relatively deep hierarchies. As a result, it is not suitable for browsing [4][15]. The other frequent item set based algorithm HFTC [4] provides a relatively flat hierarchy but its different branches of hierarchy decrease the accuracy. FIHC uses sibling merging method and overcomes the problem and it gets higher accuracy in browsing [7].

Li and Chung (2008) addressed the problem to improve the result of information retrieval for document clustering and stated that the requirement [16] of information retrieval is as follows: (1) The document model preserves the sequential relationship between words in the document, (2) Associating a meaningful label to each final Cluster is essential, (3) Overlapping between documents should be allowed, and (4) The high dimensionality of text document should be reduced. The authors refer to Fung B.C.M. et al. (2003) for the definition of document clustering with unsupervised and automatic grouping [13]

and proposed text-clustering algorithm, *Clustering Frequent Word Sequences* (CFWS) [16] where document is reduced to compact document by keeping only the frequent words to reduce high dimensionality. Experiment was conducted on a SuSE Linux PC with a Celeron 500 MHz processor and 384 MB memory using C++ and stated that CFWS algorithm is more efficient than the suffix tree clustering algorithm [17], which clusters the documents by constructing the suffix tree of all the sentences of the documents in the collection.

Frequent Term based Clustering (FTC) [4] is a text clustering technique, which uses frequent term sets and dramatically decreases the dimensionality of the document vector space [11], thus addressing itself to the problem of text clustering, namely, very high dimensionality of the data and very large size of the databases. The performance of the FTC algorithm is largely determined by the function overlap [4]. Beil F. et al. (2002) introduces two approaches, upon calculating the overlap namely, Standard overlap and Entropy overlap, and finally abandons Standard overlap due to its inherent deficiency [11]. The evaluation of FTC over bi-secting k-means and 9-secting k-means on the basis of experiments on both classical text documents and web documents shows that FTC is significantly more efficient [4]. However, FTC does not outperform other methods with respect to cluster quality [16].

III. DOCUMENT PREPROCESSING

All text clustering methods require several steps of preprocessing of the data. First, any non-textual information such as HTML tags and punctuation is removed from the documents. In general the documents are clustered, based on context matching or similarity. Mostly, the contexts of documents are represented by nouns. Based on this, the following assumptions were made in document dimension reduction [13]:

- Elimination of words which possess less than 3 characters.
- Elimination of general words
- Elimination of adverbs and adjectives
- Elimination of non-noun verbs

The following assumptions have been made to achieve frequent term generation:

- For small document, each line is treated as a record.
- For large document, each paragraph is treated as a record.

Dimension reduction improves the performance of text mining techniques to process the data with a reduced number of terms. Two improved dimension reduction algorithms namely, stemming and frequent term generations are used [13]. The morphological variant in the given document, stopping words and grammatical words are identified and removed by improved stemming algorithms [13]. The frequent term set generation algorithm generates frequent terms, which satisfy the given minimum support.

IV. TEXT CLUSTERING ATTRIBUTES SELECTION

Generally, text clustering is performed in two stages, namely, frequent term set generation (dimension reduction) and grouping of frequent term documents. Frequent term set generation is characterised by the attribute minimum support threshold and grouping of frequent term documents are characterised by matching threshold.

A. Minimum Support Threshold

In the first stage, the document database is reduced, based on the value which was selected as minimum support for the terms called minimum support threshold [1][8]. If the minimum support takes less value, the dimension reduction is less. In order to get more reduction in size the value of minimum support should be high. Suppose, a maximum value of minimum support is chosen, then the theme of the document may be reduced. So, care must be given when selecting the minimum support threshold value. In clustering aspect, if the minimum support value is low, the numbers of frequent terms are more and clustering possibility is high. For high value, the number of frequent terms are less and the clustering possibility is low [4].

B. Matching Threshold

In the second stage, the grouping of documents is carried out by finding the match of frequent terms between the documents which is measured by a value called matching threshold [8][10]. In general, matching is the ratio of number of common terms between documents to the total number of terms. For low matching threshold value, the grouping of document is more and the overlap of frequent term is more. For high matching threshold value, the grouping of document is less and the overlap of frequent term is less. The overlap of a cluster with the other clusters is the smaller for smaller value of frequent terms of its documents [4].

Problem Definition

Let $D = \{d_1, d_2, d_3, \ldots, d_n\}$ be a database of text documents and T be the set of all terms occurring in the documents of D. Each document d_i is represented by the set of terms occurring in d_i, where $d_1 = \{t_{11}, t_{12}, \ldots, t_{1m}\}$, $d_2 = \{t_{21}, t_{22}, \ldots, t_{2m}\}$ be a set of frequent term scalar vector of document d_1 and d_2 i.e., $d_i \subseteq D$ $d_j \subseteq D$ and $t_{ij} \in D$ where $i = 1 \ldots n$, and $j = 1 \ldots m$. Let min-supp be a real number, $0 \leq \text{min-supp} \leq 1$. Let $F = \{f_1, f_2, f_3, \ldots, f_k\}$ be the set of all frequent term sets in D with respect to min-supp, the set of all term sets contained in atleast min-supp of the D documents, i.e.,

$$F = \{f_i \subseteq T \parallel \text{cov}(f_i) \mid \geq \text{min-supp}.|D|\}$$

A frequent term set of cardinality k is called frequent k-term set. The cover of each element f_i of F can be regarded as a cluster. Let the clustering of D in m sets be defined as R = $\{C_1, C_2, C_3, \ldots, C_m\}$ such that, each cluster C_i contains atleast one document i.e. $C_i \neq \emptyset$; i= 1 . . . . , m. The union of all clusters is the set D:

$$\cup_{i=1}^{m} C_i = D$$

No two clusters have documents in common, i.e.,

$C_i \cap C_j = \emptyset$, $i \neq j$, i, j = 1 . . . , m.

An example represenatation of text document database (D) in terms of frequent terms (t), the documents are denoted by d is given in Table 1. It consists of five documents each of which has its own frequent terms with respect to the given minimum support to a maximum of 4 terms.

Table 1 Representation of Text Database (D) as Frequent Terms

Documents	Frequent Terms			
	t_1	t_2	t_3	t_4
d_1	A	B	C	D
d_2	A	C	F	G
d_3	F	G	I	J
d_4	D	L	M	-
d_5	A	K	X	M

V. CLUSTERING ALGORITHM

A. Min-Match Cluster Algorithm

Description

Let A and B be frequent term sets of two documents d_1 and d_2 represented as vectors. Then, the matching between the two vectors is characterized by using the assumption with reference to minimum vector, i.e., the matching is denoted by min (V_m) and is defined as the number of common elements between vector A and B to the number of elements in the minimum of the two sets, i.e.,

$$\min(V_m) = \frac{n(A \cap B)}{n\{\min\{A,B\}\}}$$

or $$\text{min\_matching} = \frac{\text{match term count}}{\min(count\{A,B\})}$$

Consider the following example:-

Let X = {A, B, C, D} and Y = {D, L, M},

$$\text{then, } \min(V_m) = \frac{1}{3} = 0.3333\ (33.33\ \%)$$

In this example, the matching term is D and gives the count as 1 out of three terms in minimum vector.

Algorithm

In this algorithm, step 2 selects a vector as a comparable vector. Steps 5 to 7 are used to find out the minimum vector from the two input vectors specified in steps 2 and 5 and assign its length as minimum vector count. In step 8, the matching terms between two vectors are calculated by using binary search concept. In step 9, matching percentage between vectors is calculated using *minimum vector count*. In step 10, the highest matching vector between the two vectors is selected and updates the value of highest match vector. The above process (steps 5 to 11) is repeated until the comparable vector selected in step 1 has to compare all the remaining vectors.

In steps 15 and 16, if the highest match vector is found, then a) Its frequent terms are added to the terms of comparable vector selected in step 2, and b) Add the highest match cluster to the comparable cluster (step 16). In steps 17 and 18, remove the highest match cluster from the cluster list (step 17). Remove the highest match cluster terms from the frequent term list (step 18). The above process (steps 2 to 19) is repeated until the vectors are compared to all other vectors in the frequent vector list.

```
D: Document database
FTL: frequent_term_list
CL: Cluster list
FT: frequent_terms

Min-Cluster(CL,FTL,D)
1. For each FT i in FTL do
2.    t1 = i^th index frequent_terms
3.    Initialise high_percent_matching = -1 and cluster_index= -1
4. For each FT j in FTL do
5.   if (i≠ j) then
          t2 = j^th index_frequent words
6.    if (t1.length < t2.length) then
            total_terms = t1.length
7.    Else
        total_terms=t2.length
      End if
8.    match= Calculate matching terms between vector i and j using
Binary Search
9.    matching_percent = match * 100 / total_terms
10.       if (matching_percent > matching_threshold) and
                ( high_percent_matching <  matching_percent) then
                  high_percent_matching = matching_percent and
cluster_index = j
11.       End if
12. End if
13. Next loop (j)
14. if (cluster_index ≠ -1) then
15.    Add frequent_term_list(cluster_index) to frequent_term_list(i)
16.   Add Cluster_list(cluster_index) to Cluster_list(i)
17.   Remove Cluster_list(cluster_index)from Cluster_list
18.   Remove frequent_term_list(cluster_index)from frequent_term_list
19. End if
20. Next loop (i)
```

Figure 1 Min-Match Cluster Algorithm

B. Max-Match Cluster Algorithm

Description

Let A and B be two frequent term sets of two documents d_1 and d_2 represented as vectors. Then, the matching between these two vectors is characterized by using the assumption with reference to the maximum vector, i.e., the matching is denoted by max (V_m) and is defined as the number of common elements between vector A and B to the number of elements in the maximum of two sets, i.e.,

$$\max\ (V_m) = \frac{n(A \cap B)}{n\{\max\{A, B\}\}}$$

or max_matching = $\frac{\text{match term count}}{\max\ (count\{A,B\})}$

Consider the following example:-

Let X = {A, B, C, D} and Y = {D, L, M},

then, max $(V_m) = \frac{1}{4} = 0.25\ (25\ \%)$

In this example, the matching term is D and gives the count as 1, out of the 4 terms in maximum vector.

Algorithm

In this algorithm, step 2 selects a vector as a comparable vector. Steps 5 to 7 are used to find out the maximum vector from the two input vectors specified in step 2 and 5 and assign its length as maximum vector count. In step 8, the matching terms between two vectors are calculated by using binary search concept. In step 9, matching percentage between vectors is calculated using *maximum vector count*.

In step 10, the highest matching vector between the two vectors is selected and the value of highest match vector is updated. The above process (steps 5 to 11) is repeated until the comparable vector selected in step 1 has to compare all the remaining vectors. In steps 15 and 16, if the highest match vector found, then, a) Its frequent terms are added to the terms of comparable vector selected in step 2, and b) Add the highest match cluster to the comparable cluster (step 16). In steps 17 and 18, remove the highest match cluster from the cluster list (step 17). Remove the highest match cluster terms from the frequent term list (step 18). The above process (steps 2 to 19) is repeated until the vectors are compared to all other vectors in the frequent vector list.

```
D: document database
FTL: frequent_term_list
CL: Cluster list
FT: frequent_terms

Max-Cluster(CL,FTL,D)
1. For each FT i in FTL do
2.    t1 = i^th index frequent_words
3.    Initialise high_percent_matching = -1 and cluster_index= -1
4. For each FT j in FTL do
5.   if (i≠ j) then
                  t2 = j^th index_frequent words
6.    if (t1.length < t2.length) then
                  total_terms = t2.length
7.    Else
              total_terms=t1.length
        End if
8. match= Calculate matching terms between vector i and j using
Binary Search
9. matching_percent = match * 100 / total_terms
10.        if (matching_percent > matching_threshold) and
                  ( high_percent_matching <  matching_percent) then
                    high_percent_matching = matching_percent and
cluster_index = j
11.       End if
12. End if
13. Next loop (j)
14.  if (cluster_index ≠ -1) then
15.   Add frequent_term_list(cluster_index) to frequent_term_list(i)
16.   Add Cluster_list(cluster_index) to Cluster_list(i)
17.   Remove Cluster_list(cluster_index)from Cluster_list
18.   Remove frequent_term_list(cluster_index)from frequent_term_list
19. End if
20. Next loop (i)
```

Figure 2 Max-Match Cluster Algorithm

C. Min-Max Match Cluster Algorithm

Description

Let A and B be two frequent term sets of two documents d_1 and d_2 represented as vectors. Then the matching between these two vectors is characterized by using the assumption with reference to the maximum vector, i.e., the matching is denoted by min_max (V_m) and is defined as the number of matching terms multiplied by 2 to the number of elements of two sets, i.e.,

$$\min\ \_\max\ (V_m) = \frac{n(A \cap B)}{n\{A, B\}}$$

or min _max_matching = $\frac{\text{match term count * no. of vector}}{(count\{A,B\})}$

Consider the following example:-

Let X = {A, B, C, D} and Y = {D, L, M},

then, max $(V_m) = \frac{1 * 2}{7} = \frac{2}{7} = 0.285\ (28.5\ \%)$

In this example, the matching term D gives the count as 1 in both vectors out of 7 terms, the total count of the two vectors.

Algorithm

In this algorithm, step 2 selects a vector as a comparable vector. Steps 5 to 7 are used to find out the total length of the two input vectors specified in steps 2 and 5 to assign as min-max vector count. In step 8, the matching terms between two vectors are calculated by using binary search concept. In step 9, matching percentage between vectors is calculated, using *min-max vector count*. In step 10, the highest matching vector between the two vectors is selected and updates the value of highest match vector. The above process (steps 5 to 11) is repeated until the comparable vector selected in step 1 has to compare all the remaining vectors.

```
D: document database
FTL: frequent_term_list (set contains set of Frequent Terms)
CL: Cluster list (set contains set of Input Files Names)
FT: frequent_terms
t1, t2: Frequent Term Set

Min-MaxCluster (CL,FTL,D)

1.  For each FT i in FTL do
2.    t1 = i^th index frequent_words
3.    Initialise high_percent_matching = -1 and cluster_index= -1
4.   For each FT j in FTL do
5.    if (i≠ j) then
                  t2 = j^th index_frequent words
6.      t3 = i^th FTL UNION j^th FTL
7.      total_terms = t3.length
8.      match= Calculate matching terms between vector i and j using Binary Search
9.      matching_percent = match * 2* 100 / total_terms
10.      if (matching_percent > matching_threshold) and
                ( high_percent_matching <  matching_percent) then
                   high_percent_matching = matching_percent and cluster_index = j
11.     End if
12.    End if
13.  Next loop (j)
14.  if (cluster_index ≠ -1) then
15.   Add frequent_term_list(cluster_index) to frequent_term_list(i)
16.  Add Cluster_list(cluster_index) to Cluster_list(i)
17.    End if
18.  Remove Cluster_list(cluster_index)from Cluster_list
19.  Remove frequent_term_list(cluster_index)from frequent_term_list
20. Next loop (i)
```

Figure 3: Min-Max-Match Cluster Algorithm

In steps 15 and 16, if the highest match vector found then a) Its frequent terms are added to the terms of comparable vector selected in step 2, and b) Add the highest match cluster to the comparable cluster (step 16). In steps 17 and 18, remove the highest match cluster from the cluster list (step 17). Remove the highest match cluster terms from the frequent term list (step 18). The above process (steps 2 to 19) is repeated until the vectors are compared to all other vectors in the frequent vector list.

D. Cosine Similarity Cluster Algorithm

Description

Let A and B be two frequent term sets of two documents A and B represented in the vector space model. Then the matching between these two vectors is characterized by using the concept with reference to the cosine angle between the two vectors. The matching is denoted by cosine (V_m). The cosine similarity is defined by the cosine angle between the two vectors, i.e.,

$$Cosine\ (V_m) = \frac{(A.B)}{||A||.||B||}$$

where . denotes the vector dot product and || denotes the length of vector

Table 2 gives an example of representation of text document database D in terms of vector space model (t) and the documents are denoted by d. In this model, the terms are denoted by a binary value 0 and 1. A 0 indicates that the term is infrequent and a 1 indicates that the term is frequent.

Table 2 Representation of Text Database D as Frequent Vector Space Model

Documents	Frequent Terms					
	t_1	t_2	t_3	t_4	t_5	t_6
d_1	1	1	1	0	1	0
d_2	0	0	1	1	0	1
d_3	1	0	0	1	0	1
d_4	1	1	0	0	1	0
d_5	0	0	1	1	1	1

Consider the following example

Let X = {A, B, C, D} and Y = {D, L, M}

The corresponding vector space representation is

X = {1, 1, 1, 1, 0, 0} and Y = {0, 0, 0, 1, 1, 1}

then Cosine $(V_m) = \frac{1}{6} = 0.166\ (16.6\ \%)$

In this example, the dot product of the two vectors gives the count as 1, out of the vector length 6.

Algorithm

In this algorithm steps 4 to 6 are used to find out the total terms of the similarity vector generated from the given two input vectors. In step 5, the similarity vector is formed by taking the union of the two given vectors. The steps 7 and 8 are used to generate binary vector for the input vectors respectively. In step 9, the matching between the two vectors is calculated by taking the dot product of vector 1 and vector 2. In step 10, the matching percentage is calculated using cosine similarity formula. In step 11, the highest matching vector is selected and updates the value of high match vector. The above process (steps 4 to 14) is repeated until the comparable vector has to compare with all the remaining vectors.

```
D: document database
FTL: frequent_term_list
CL: Cluster list
FT: frequent_terms

COSINE-SIMILARITY Cluster(CL,FTL,D)
1. For each FT i in FTL do
2.    Initialise high_percent_matching = -1 and cluster_index= -1
3.   For each FT j in FTL do
4.     if (i≠ j) then
5.        Similarity_Vector = i^th FTL  UNION  j^th FTL
6.        total_terms = Similarity_Vector. length;
7.        Vector1 = GetVector (Similarity_Vector, i^th cluster FTL)
8.        Vector2 = GetVector (Similarity_Vector, j^th cluster FTL)
9.       Match = dot product of vector 1.  Vector2
10.       Matching_percent = No_of_Matching / total_terms
11.       if (Matching_percent > matching_threshold) and
                  (high_percent_matching < Matching_percent) then
                     high_percent_matching = Matching_percent and cluster_index = j
12.      End if
13.     Next loop (j)
14.   End if
15. if (cluster_index ≠ -1) then
16.   Add frequent_term_list (cluster_index) to frequent_term_list (i)
17.   Add Cluster_list (cluster_index) to Cluster_list (i)
18.   Remove Cluster_list (cluster_index) from Cluster_list
19.   Remove frequent_term_list (cluster_index) from frequent_term_list
20. End if
21. Next loop (i)

GetVector(Similarity_Vector, FTL)
1. Vector.length = Similarity_Vector.length
2.       For each TERM in Similarity_Vector
3. if (Frequent_word_list contains TERM)
4.    Add 1 to vector
5.       Else  Add 0 to vector
6. return vector
```

Figure 4: Cosine Similarity Cluster Algorithm

In steps 16 and 17, if the highest matching vector is found, then a) Its frequent terms are added to the terms of comparable vector, and b) Add the highest match cluster to the comparable cluster (step 17). In steps 18 and 19, remove the highest match cluster from the cluster list (step 18). Remove the highest match cluster terms from frequent term list (step 19). The above process (steps 2 to 20) is repeated until the vectors are compared to all other vectors in the frequent vector list.

The getvector( ) procedure is used to generate binary vector from a given vector. Step 1 is used to determine the binary vector length. In steps 3 to 5, binary value 1 or 0 is added to the vector depends on the matching of the term with the frequent word list. Add 1 to the vector if matched (step 4) or else add 0 to the vector (step 5). The above process is repeated to add binary value for all the terms in the given frequent term vector.

VI. RESULTS AND DISCUSSIONS

Table 3 represents the combined F-measures of FTC algorithm along with the proposed four algorithms for Reuters Transcribed Subset (RTS) - 200 documents-249 KB. The F-measure chart for five algorithms of Reuters Transcribed subset in Figure 5 shows that the proposed four algorithms show better cluster quality than FTC.

The minimum support of frequent terms is usually in the range of 5-15%. When the minimum support is too large, the total number of frequent terms would be very small, so that the resulting document would not have enough information about the original data set. But this may depend on data set.

Table 3 F-measure Value for Reuters Transcribed Subset

Minimum Support	FTC	Min-Match	Max-Match	Min-Max-Match	Cosine Similarity
5	0.327	0.585	0.779	0.747	0.821
15	0.399	0.606	0.779	0.754	0.821
25	0.404	0.622	0.779	0.744	0.817
35	0.427	0.634	0.779	0.744	0.818
45	0.497	0.627	0.795	0.745	0.833
55	0.548	0.627	0.800	0.750	0.823
65	0.646	0.690	0.800	0.780	0.832
75	0.756	0.728	0.829	0.807	0.840
85	0.782	0.762	0.861	0.832	0.865
95	0.782	0.786	0.900	0.845	0.875

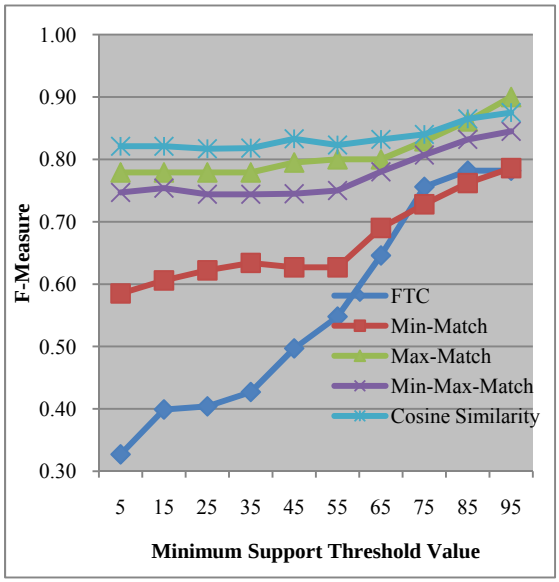

Figure 5 F-measure Chart for Reuters Transcribed Subset

Regarding cluster quality, among the four algorithms, Cosine Similarity algorithm is first, Max-Match algorithm is second, Min-Max-Match algorithm is third and Min-Match algorithm is fourth.

Paired t-test is employed to compare the FTC algorithm and the developed four algorithms based on F-measure value with respect to minimum support. FTC algorithm and each of the developed clustering algorithms have been tested by applying paired t-test to find out the significant difference in F-measures between FTC and the developed algorithms. The results are shown in the Table 4. It is observed the t-value indicates that there is a difference between FTC and Max-Match, Min-Max-Match and Cosine Similarity clustering algorithms based on F-measure with respect to minimum support is significant at 5% level and Min-Match clustering algorithm is significant at 1% level.

Table 4 Paired t-test between Algorithms based on F-measure for RTS Sample

Comparison of Algorithms	Method	Mean	Standard Deviation	Mean Difference	t-value	Significance
FTC Vs Min	FTC	0.557	0.173	0.110	3.213	*
	Min	0.667	0.070			
FTC Vs Max	FTC	0.557	0.173	0.253	5.775	**
	Max	0.810	0.041			
FTC Vs Min-Max	FTC	0.557	0.173	0.218	4.978	**
	Min-Max	0.775	0.039			
FTC Vs Similarity	FTC	0.557	0.173	0.278	5.615	**
	Similarity	0.835	0.020			

*- Significant at 5% level
** - Significant at 1% level

Table 5 represents the performance evaluation of FTC algorithm and the proposed Min-Match, Max-Match, Min-Max-Match and Cosine Similarity algorithms for the different data sets, where S1 denotes Reuters Transcribed subset(RTS)- 200 documents-249 KB, S2 denotes Mini-News Group- 200 documents-547 KB, S3 denotes Mini-News Group-700 documents-1915 KB, S4 denotes Mini-News Group-1000 documents- 2591 KB, S5 denotes PDF-54 documentss-2999 KB, and S6 denotes Reuters-21578-22 files-20275 KB.

Table 5 Performance of Developed Algorithms for Different Data Sets

Data sets	File Size (KB)	Execution Time in Seconds				
		FTC	Min-Match	Max-Match	Min-Max-Match	Cosine Similarity
S1	249	13.25	12.44	12.36	12.08	12.07
S2	547	127.41	38.39	67.49	38.38	37.30
S3	1915	222.43	195.59	157.81	163.14	155.77
S4	2591	426.47	233.41	260.70	195.80	153.72
S5	2999	1698.72	404.92	468.83	392.75	236.37
S6	20275	2354.88	796.42	795.58	810.17	611.97

Figure 6 shows the performance of developed algorithms for six different data sets and shows that execution time increases as and when the document data size increases. It is observed that the execution time of FTC algorithm is higher than the other four developed algorithms.

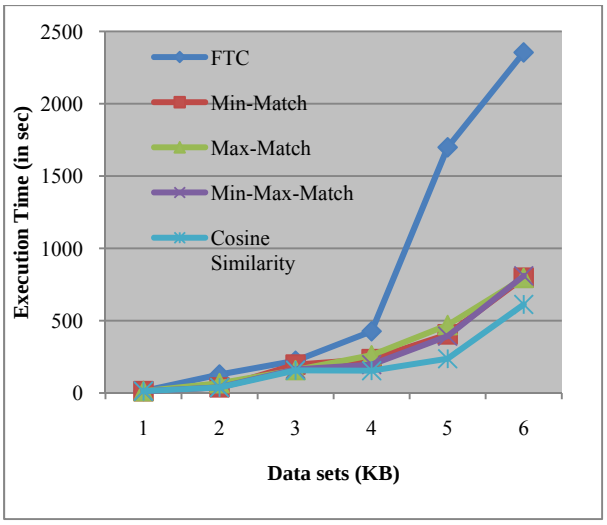

Figure 6 Performance of Developed Algorithms for Different Data Sets

VII. CONCLUSIONS

In this paper, the significance of text clustering attributes namely minimum support threshold and matching threshold are discussed. For effective text clustering, four new clustering algorithms were proposed. All the four algorithms are compared with the standard FTC algorithm to show their competency. The F-measure chart of clustering algorithm for RTS sample and the corresponding table show that the developed four algorithms have higher F-measure value and perform better cluster quality than FTC algorithm. The performance of algorithms for six different data sets shows that the developed algorithms perform better than FTC. In the entire newly developed and implemented clustering algorithm, the performance results are encouraging.

REFERENCES

[1] Agrawal R., and Srikant R., "Fast algorithm for mining association rules," Proceedings of 20th International Conference on Very Large Data Bases, VLDB 94, Santiago de Chile, Chile, 1994, pp.487-499.

[2] Ahonen H., Myka H.," Mining all Maximal Frequent Word Sequences in a Set of Sentences," Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 255-256, 2005.

[3] Allan J., "HARD Track Overview in TREC High Accuracy, Retrieval from Documents, " Proceedings of the 12th Text Retrieval Conference, 2003, pp. 24-37.

[4] Beil F., Ester M. and Xu X., "Frequent Term-based Text Clustering," Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, 436-442.

[5] Dubes R. C. and Jain A. K., "Algorithms for Clustering Data," Prentice Hall, Englewood Cliffs NJ, U.S.A, 1988.

[6] Frakes B. and Baeza-Yates R.," Information Retrieval: Data Structures and Algorithms," Englewood Cliffs, N.J.: Prentice Hall, 1992.

[7] Fung B.C.M., Wang K. and Ester M., "Hierarchical Document Clustering using Frequent Item sets." Proceedings of SIAM International Conference on Data Mining, 2003, 180-304.

[8] Han J., Kamber M.,"Data Mining: Concepts and Techniques," Morgan Kaufmann (Elsevier), 2006.

[9] Kaufman L. and Rousseeuw P.J., "Finding Groups in Data: An Introduction to Cluster Analysis," John wiley and Sons, March 1990.

[10] Khaled B. Shaban.," A Semantic Approach for Document Clustering," Journal of software. VOL 4. NO 5, July 2009.

[11] Liu X., He P. and Wang H., "The Research of Text Clustering Algorithms based on Frequent Term Sets," Proceedings of the 4th International Conference on Machine Learning and Cybernetics, pp. 2352- 2356, August 2005.

[12]Manning C.D., Raghavan P. and Schutze H., "Introduction to Information Retrieval," Cambridge University Press, Cambridge, UK, 2008.

[13]Ponmuthuramalingam.P. and Devi.D., "Effective Dimension Reduction Techniques for Text Documents," International Journal on Computer Science and Network Security(ICSNS), Vol 10, No. 7, 2010.

[14]Sebastiani F., "Machine Learning in Automated Text Categorization," ACM Computing Surveys, Vol. 34, No. 1, pp. 1-47, 2002.

[15] Steinbach M., Karypsis G. and Kumar V., "A Comparison of Document Clustering Techniques, "KDD-2000 Workshop on Text Mining, 2000, 203-215.

[16]Yanjun Li, Soon M. Chung, John D. Holt," Text Document clustering based on frequent word meaning sequences," In Data & Knowledge Engineering 2008 pp. 381-404.

[17] Zamir O. and Etzioni O., "Web Document Clustering: A Feasibility Demonstration," Proceedings of Annual ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, 46-54.

P.Ponmuthuramalingam received his Masters Degree in Computer Science from Alagappa University,Karaikudi in 1988 and the M.Phil in Computer Science from Bharathidasan University, Tiruchirapalli. He is working as Associate Professor in Computer Science, Government Arts College, Coimbatore since 1989. His research interest includes Text mining, Semantic Web, Network Security and Parallel Algorithms .

T.Devi received the Master of Computer Applications from P.S.G. College of Technology, Coimbatore in 1987 and Ph.D from the University of Warwick, United Kingdom in 1998. She is presently heading Department of Computer Application, School of Computer Science Engineering,Bharathiar University, Coimbatore. Prior to joining Bharathiar University, she was an Associate Professor in Indian Institute of Foreign Trade, New Delhi. Her current research centered on the Software Engineering, Product Introduction, Technical Process Management and Concurrent Engineering. She has contributed more than 60 papers in various National / International / conference/ Seminars /Symposia.

This page is left blank intentionally