Document Clustering with Grouping and Chaining Algorithms

Yllias Chali and Soufiane Noureddine

Department of Computer Science,
University of Lethbridge

Abstract. Document clustering has many uses in natural language tools and applications. For instance, summarizing sets of documents that all describe the same event requires first identifying and grouping those documents talking about the same event. Document clustering involves dividing a set of documents into non-overlapping clusters. In this paper, we present two document clustering algorithms: *grouping algorithm*, and *chaining algorithm*. We compared them with k-means and the EM algorithms. The evaluation results showed that our two algorithms perform better than the k-means and EM algorithms in different experiments.

1 Introduction

Document clustering has many uses in natural language tools and applications. For instance, summarizing sets of documents that all describe the same event requires first identifying and grouping those documents talking about the same event. Document clustering involves dividing a set of texts into non-overlapping clusters, where documents in a cluster are more similar to one another than to documents in other clusters. The term more *similar*, when applied to clustered documents, usually means closer by some measure of proximity or similarity.

According to Manning and Schutze [1], there are two types of structures produced by clustering algorithms, hierarchical clustering and flat or non-hierarchical clustering. Flat clustering are simply groupings of similar objects. Hierarchical clustering is a tree of subclasses which represent the cluster that contains all the objects of its descendants. The leaves of the tree are the individual objects of the clustered set. In our experiments, we used the non-hierarchical clustering k-means and EM [2] and our own clustering algorithms.

There are several similarity measures to help find out groups of related documents in a set of documents [3]. We use identical word method and semantic relation method to assign a similarity score to each pair of compared texts. For the identical word method, we use k-means algorithm, the EM algorithm, and our own *grouping algorithm* to cluster the documents. For the semantic relation method, we use our own *grouping algorithm* and *chaining algorithm* to do the clustering job. We choose WordNet 1.6 as our background knowledge. WordNet consists of synsets gathered in a hypernym/hyponym hierarchy [4]. We use it to get word senses and to evaluate the semantic relations between word senses.

2 Identical Word Similarity

To prepare the texts for the clustering process using identical word similarity, we perform the following steps on each of the selected raw texts:

1. Preprocessing which consists in extracting file contents from the raw texts, stripping special characters and numbers, converting all words to lower cases and removing stopwords, and converting all plural forms to singular forms.
2. Create document word vectors: each document was processed to record the unique words and their frequencies. We built the local word vector for each document, each vector entry will record a single word and its frequency. We also keep track of the unique words in the whole texts to be tested. After processing all the documents, we convert each local vector to a global vector using the overall unique words.
3. Compute the identical word similarity score among documents: given any two documents, if we have their global vectors $\boldsymbol{x}$, $\boldsymbol{y}$, we can use the cosine measure [5] to calculate the identical word similarity score between these two texts.

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-dimensional vectors in a real-valued space.

Now, we determined a global vector for each text. We also have the identical word similarity scores among all texts. We can directly use these global vectors to run the k-means or the EM algorithms to cluster the texts. We can also use the identical word similarity scores to run *grouping algorithm* (defined later) to do the clustering via a different approach.

3 Semantic Relation Similarity

To prepare the texts for clustering process using semantic relation similarity, the following steps are performed on each raw texts:

1. Preprocessing which consists in extracting file contents, and removing special characters and numbers.
2. Extract all the nouns from the text using part-of-speech tagger (i.e. UPennsylvania tagger). The tagger parses each sentence of the input text into several forms with specific tags. We get four kinds of nouns as the results of running the tagger: NN, NNS, NNP and NNPS. We then run a process to group all the nouns into meaningful nouns and non-meaningful nouns. The basic idea is to construct the largest compound words using the possible adjective and following nouns, then check whether or not the compound words have a meaning in WordNet. If not, we break the compound words into possible smaller ones, then check again until we find the ones with meanings in

Wordnet. When we get a noun (or a compound noun) existing in WordNet, we insert it into the meaningful word set, which we call set of regular nouns, otherwise we insert it into the non-meaningful word set, which we call set of proper nouns.

During the processing of each document, we save the over-all unique meaningful nouns in an over-all regular nouns set. Because of the big overhead related to accessing WordNet, we try to reduce the overall access times to a minimal level. Our approach is to use these over-all unique nouns to retrieve the relevant information from WordNet and save them in a global file. For each sense of each unique noun, we save its synonyms, two level hypernyms, and one level hyponyms. If any process frequently needs the WordNet information, it can use the global file to store the information in a hash and thus provides fast access to its members.

3. Word sense disambiguation.

   Similarly to Galley and McKeown [6], we use lexical chain approach to disambiguate the nouns in the regular nouns for each document [7,8]. A lexical chain is a sequence of related words in the text, spanning short (adjacent words or sentences) or long distances (entire text). WordNet is one lexical resource that may be used for the identification of lexical chains. Lexical chains can be constructed in a source text by grouping (chaining) sets of word-senses that are semantically related. We designate the following nine semantic relations between two senses:

   (a) Two noun instances are identical, and used in the same sense;
   (b) Two noun instances are used in the same sense (i.e., are synonyms);
   (c) The senses of two noun instances have a hypernym/hyponym relation between them;
   (d) The senses of two noun instances are siblings in the hypernym/hyponym tree;
   (e) The senses of two noun instances have a grandparent/grandchild relation in the hypernym/hyponym tree;
   (f) The senses of two noun instances have a uncle/nephew relation in the hypernym/hyponym tree;
   (g) The senses of two noun instances are cousins in the hypernym/hyponym tree (i.e., two senses share the same grandparent in the hypernym tree of WordNet);
   (h) The senses of two noun instances have a great-grandparent/great-grandchild relation in the hypernym/hyponym tree (i.e., one sense's grandparent is another sense's hyponym's great-grandparent in the hypernym tree of WordNet).
   (i) The senses of two noun instances do not have any semantic relation.

   To disambiguate all the nouns in the regular nouns of a text, we proceed with the following major steps:

   (a) Evaluate the semantic relation between any two possible senses according to the hypernym/hyponym tree in WordNet. For our experiments, we use the following scoring scheme for the relations defined above as shown in Table 1. The score between A_i (sense i of word A) and B_j (sense j of word B) is denoted as $score(A_i, B_j)$. These scores are established empirically and give more weight to closer words according to WordNet hierarchy.

Table 1. Scoring Scheme for Relations

Relation	$Score(A_i, B_j)$
Identical	log(16)
Synonyms	log(15)
Hypernyms/hyponyms	log(14)
Siblings	log(13)
Grandparent/grandchild	log(12)
Uncle/nephew	log(11)
Cousins	log(10)
Great-grandparent/great-grandchild	log(9)
No relation	0

(b) Build the lexical chains using all possible senses of all nouns. To build the lexical chains, we assume each noun possesses all the possible senses from WordNet. For each sense of each noun in a text, if it is related to all the senses of any existing chain, then we put this sense into this chain, else we create a new chain and push this sense into the new empty chain. After this, we will have several lexical chains with their own scores.

(c) Using the lexical chain, try to assign a specific sense to each nouns. We sort the chains by their scores in a non-increasing order. We select the chain with the highest score and assign the senses in that chain to the corresponding words. These words are disambiguated now. Next, we process the next chain with the next highest score. If it contains a different sense of any disambiguated words, we skip it to process the next chain until we reach the chains with a single entry. We mark the chains which we used to assign senses to words as selected. For the single entry chains, if the sense is the only sense of the word, we mark it as disambiguated. For each undisambiguated word, we check each of its senses against all the selected chains. If it has a relation with all the senses in a selected chain, we will then remember which sense-chain pair has the highest relation score, then we assign that sense to the corresponding word.

After these steps, the leftover nouns will be the undisambiguated words. We save the disambiguated words and the undisambiguated words with their frequencies for calculating the semantic relation scores between texts.

4. Compute the similarity score for each pair of texts.

Now, we should have three parts of nouns for each text: disambiguated nouns, undisambiguated nouns and the non-meaningful nouns (proper nouns). We will use all of them to calculate the semantic similarity scores between each pair of texts. For the purpose of calculating the semantic similarity scores among texts, we use only the first three relations (a), (b), and (c) and the last relation (i) and their corresponding scores defined in *Table 1*. For a given text pair, we proceed as in the following steps to calculate the similarity scores:

– Using the disambiguated nouns, the score $score_1$ of the similarity between two texts T_1 and T_2 is computed as follows:

$$score_1 = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} score(A_i, B_j) \times freq(A_i) \times freq(B_j)}{\sqrt{\sum_{i=1}^{n} freq^2(A_i)}\sqrt{\sum_{j=1}^{m} freq^2(B_j)}} \tag{2}$$

where A_i is a word sense from T_1 and B_j is a word sense from T_2; $score(A_i, B_j)$ is a semantic relation score defined in *Table 1*; n and m are the numbers of disambiguated nouns in T_1 and T_2; $freq(x)$ is the frequency of a word sense x.

– For the undisambiguated nouns, if two nouns are identical in their word formats, then the probability that they take the same sense in both texts is $1/s$, where s is the number of their total possible senses. The similarity score $score_2$ between two texts T_1 and T_2 according to the undisambiguated nouns is computed as follows:

$$score_2 = \frac{\sum_{i=1}^{n} \frac{log(16) \times freq_1(A_i) \times freq_2(A_i)}{s_i}}{\sqrt{\sum_{i=1}^{n} freq_1^2(A_i)}\sqrt{\sum_{j=1}^{n} freq_2^2(A_j)}} \tag{3}$$

where A_i is a word common to T_1 and T_2; n is the number of common words to T_1 and T_2; $freq_1(A_i)$ is the frequency of A_i in T_1; $freq_2(A_i)$ is the frequency of A_i in T_2; s_i is the number of senses of A_i.

– The proper nouns are playing an important role in relating texts to each other. So, we use a higher score (i.e., $log(30)$) for the identical proper nouns. The similarity score $score_3$ between two texts T_1 and T_2 among the proper nouns between is computed as follows:

$$score_3 = \frac{\sum_{i=1}^{n} log(30) \times freq_1(A_i) \times freq_2(A_i)}{\sqrt{\sum_{i=1}^{n} freq_1^2(A_i)}\sqrt{\sum_{j=1}^{n} freq_2^2(A_j)}} \tag{4}$$

where A_i is a proper noun common to T_1 and T_2; n is the number of common proper nouns to T_1 and T_2; $freq_1(A_i)$ is the frequency of A_i in T_1; $freq_2(A_i)$ is the frequency of A_i in T_2.

– Adding all the scores together as the total similarity score of the text pair:

$$score = score_1 + score_2 + score_3 \tag{5}$$

Now we make it ready to use the *grouping algorithm* or *chaining algorithm* defined shortly to cluster the texts.

4 Clustering Algorithms

Generally, every text should have a higher semantic similarity score with the texts from its group than the texts from a different groups [9]. There are a few rare cases where this assumption could fail. One case is that the semantic similarity score does not reflect the relationships among the texts. Another case is that the groups are not well grouped by common used criteria or the topic is too broad in that group.By all means, the texts of any well formed clusters should have stronger relations among its members than the texts in other clusters. Based on this idea, we developed two text clustering algorithms: *grouping algorithm* and *chaining algorithm* . They share some common features but with different approaches.

One major issue in partitioning texts into different clusters is choosing the cutoff on the relation scores. Virtually, all texts are related with each other to some extent. The problem here is how similar (or close) they should be so that we can put them into one cluster and how dissimilar (or far away) they should be so that we can group them into different clusters. Unless the similarity scores among all the texts can be represented as binary values, we will always face this problem with any kind of texts. In order to address this problem, we introduce two reference values in our text clustering algorithms: *high-threshold* and *low-threshold.* The high-threshold means the high standard for bringing two texts into the same cluster. The low-threshold means the minimal standard for possibly bringing two texts into the same cluster. If the score between any two texts reaches or surpasses the high-threshold, then they will go to the same cluster. If the score reaches the low-threshold but is lower than the high-threshold, then we will carry out further checking to decide if we should bring two texts into the same cluster or not, else, the two texts will not go to the same cluster.

We get our high-threshold and low-threshold for our different algorithms by running some experiments using the grouped text data. The high-threshold we used for our two algorithms is 1.0 and the low-threshold we used is 0.6. For our experiment, we always take a number of grouped texts and mix them up to make a testing text set. So, each text must belong to one cluster with certain number of texts.

4.1 Grouping Algorithm

The basic idea is that each text could gather its most related texts to form an initial group, then we decide which groups have more strength over other groups, make the stronger groups as final clusters, and use them to bring any possible texts to their clusters. First, we use each text as a leading text (T_l) to form a cluster. To do this, we put all the texts which have a score greater than the high-threshold with T_l into one group and add each score to the group's total score. By doing this for all texts, we will have N possible different groups with different entries and group scores, where N is the number of the total texts in the set. Next, we select the final clusters from those N groups. We arrange all the groups by their scores in a non-increasing order. We choose the group with the highest score and check if any text in this group has been clustered to the existing final clusters or not. If not more than 2 texts are overlapping with the final clusters, then we take this group as a final cluster, and remove the overlapping texts from other final clusters. We process the group with the next highest score in the same way until the groups' entries are less than 4. For those groups, we would first try to insert their texts into the existing final clusters if they can fit in one of them. Otherwise, we will let them go to the leftover cluster which holds all the texts that do not belong to any final clusters. The following is the pseudocode for the grouping algorithm:

```
Grouping Algorithm
// Get the initial clusters
for each text $t_i$
   construct a text cluster including all the texts($t_j$)
   which score($t_i$, $t_j$) >= high-threshold;
   compute the total score of the text cluster;
   find out its neighbor with maximum relation score;
end for

// Build the final clusters
sort the clusters by their total score in non-increasing order;
for each cluster $g_i$ in the sorted clusters
   if member($g_i$) > 3 and overlap-mem($g_i$) <= 2
     take $g_i$ as a final cluster $c_i$;
     mark all the texts in $c_i$ as clustered;
   else
     skip to process next cluster;
   end if
end for

// Process the leftover texts and insert them into one of the final clusters
for each text $t_j$
   if $t_j$ has not been clustered
     find cluster $c_i$ with the highest score($c_i$, $t_j$);
     if the average-score($c_i$, $t_j$) >= low-threshold
       put $t_j$ into the cluster $c_i$;
     else if the max score neighbor $t_m$ of $t_j$ is in $c_k$
       put $t_j$ into cluster $c_k$;
     else
       put $t_j$ into the final leftover cluster;
     end if
   end if
end for

output the final clusters and the final leftover cluster;
```

where: member(g_i) is the number of members in group g_i; overlap-mem(g_i) is the number of members that are overlapped with any final clusters; score(c_i, t_j) is the sum of scores between t_j and each text in c_i; average-score(c_i, t_j) is score(c_i, t_j) divide by the number of texts in c_i.

4.2 Chaining Algorithm

This algorithm is based on the observation of the similarities among the texts in groups. Within a text group, not all texts are always strongly related with any other texts. Sometimes there are several subgroups existing in a single group, i.e., certain texts have stronger relations with their subgroup members and have a weaker relation with other subgroup members. Usually one or more texts have stronger relation crossing different subgroups to connect them together, otherwise all the texts in the group could not be grouped together. So, there is a chaining effect in each group connecting subgroups together to form one entire group.

We use this chaining idea in the *chaining algorithm*. First, for each text T_j, we find all the texts which have similarity scores that are greater or equal than the high-threshold with T_j and use them to form a closer-text-set. All the texts in that set are called closer-text of T_j.

Next, for each text which has not been assigned to a final chain, we use its initial closer-text-set members to form a new chain. For each of the texts in the chain, if any of its closer-texts are relatively related (i.e., the score >= low-threshold) to all the texts in the chain, then we add it into the current chain. One thing needs to be noticed here is that we do not want to simply bring all the closer-texts of each current chain's members into the chain. The reason is to eliminate the unwanted over-chaining effect that could bring many other texts which are only related to one text in the existing chain. So, we check each candidate text against all the texts in the chain to prevent the over-chaining effect. We repeat this until the chain's size are not increasing. If the chain has less than 4 members, we will not use this chain for a final cluster and try to re-assign the chain members to other chains.

After the above process, if any text has not been assigned to a chain we check it against all existing chains and find the chain which has highest similarity score between the chain and this text. If the average similarity score with each chain members is over low-threshold, we insert this text into that chain, else we put it into the final leftover chain. The following is the pseudocode for the chaining algorithm:

5 Application

We chose as our input data the documents sets used in the Document Understanding Conferences [10,11], organized by NIST. We collected 60 test document directories for our experiments. Each directory is about a specific topic and has about 10 texts and each text has about 1000 words. Our experiment is to mix up the 60 directories and try to reconstitute them using one of our clustering

Chaining Algorithm

```
// construct a closer-text-set for each text
for each text $t_i$ $0 < i <= N$
   for each text $t_j$ $0 < j <= N$
     if score($t_i$, $t_j$) >= high-threshold
         put $t_j$ into closer-text-set $s_i$;
     end if
   end for
end for

// Build the chains
c = 0;
for each text $t_i$ of all the texts
   if it has not been chained in
     put text $t_i$ into chain c and mark it as been chained;
     bring all the text in closer_text-set $s_i$ into the new chain c;
     mark $s_i$ as processed;
     while (the size of chain c is changing)
       for each text $t_k$ in chain c
         for each text $t_m$ in $s_k$ of $t_k$
           if the score between $t_m$ and any text in chain c >= low-threshold
             put $t_m$ into chain c;
             mark $t_m$ as been chained to chain c;
           end if
         end for
       end for
     end while
     if the size of chain c < 4
       discard chain c;
       remark the texts in chain c as unchained;
     end if
     c++;
   end if
end for

// Process the leftover texts and insert them into one of the existing chains
for each unchained text $t_j$
   find chain $c_i$ with the highest score($c_i$, $t_j$);
   if the average-score($c_i$, $t_j$) >= low-threshold
     put $t_j$ into the chain $c_i$;
   else
     put $t_j$ into the final leftover chain;
   end if
end for

output the valid chains and the final leftover chain.
```

algorithm. Then, we measure how successful are these algorithms in reconstituting the original directories. We implemented the *k-means algorithm* and the *EM algorithm* to compare them with our algorithms.

In our test, we found out that the chaining algorithm did not work well for identical method. We tested grouping algorithm, chaining algorithm, and EM algorithm with semantic method, and k-means algorithm, EM algorithm, and grouping algorithm with identical methods. We run the k-means and the EM algorithms 4 times with each experiment texts set and take the average performance. As we described before, semantic method represents text relations with scores, so k-means algorithm which needs input data in vector format will not be applied to semantic method.

6 Evaluation

For our testing, we need to compare the system clusters with the testing clusters (original text directories) to evaluate the performance of each system. We first compare each system cluster with all of the testing clusters to find the best matched cluster pair with the maximum number of identical texts. We then use recall, precision, and F-value to evaluate each matching pair. Finally, we use the average F-value to evaluate the whole system performance. For a best matched pair TC_j (testing cluster) and SC_i (system cluster), the recall (R), precision (P), and F-value (F) are defined as follows:

$$R = \frac{m}{t} \tag{6}$$

$$P = \frac{m}{m+n} \tag{7}$$

$$F(TC_j, SC_i) = \frac{2PR}{P+R} \tag{8}$$

where m is the number of the overlapping texts between TC_j and SC_i; n is the number of the non-overlapping texts in SC_i; t is the total number of texts in TC_j.

For the whole system evaluation, we use the *Average_F* which is calculated using the F-values of each matched pair of clusters.

$$Average\_F = \frac{\sum_{i,j} max(F(SCi, TCj))}{max(m,n)} \tag{9}$$

Where $i <= min(m,n)$, $j <= m$, m is the number of testing clusters, and n is the number of system clusters.

7 Results

The performance of grouping algorithm and chaining algorithm are very close using the semantic relation approach and most of their *Average_F* are over 90%. For the identical word approach, the grouping algorithm performance is much better than the performances of the k-means algorithm and the EM algorithm. The poor performance of the k-means algorithm results from randomly selected k initial values. Those initial N-dimensional values usually do not represent the whole data very well. For the semantic relation approach, both grouping and chaining algorithms performed better than the EM algorithm.

Table 2 and 3 are the system *Average_F* values for the different algorithms. The identical word similarity method used grouping algorithm, k-means algorithm, and EM algorithm. The semantic similarity method used grouping algorithm, chaining algorithm and EM algorithm.

Table 2. Comparisons of F-value using Identical Word Similarity

Identical Word Similarity		
Grouping	EM	k-means
0.98	0.81	0.66

Table 3. Comparisons of F-value using Semantic Relation Similarity

Semantic Relation Similarity		
Grouping	Chaining	EM
0.92	0.91	0.76

8 Conclusion

Document clustering is an important tool for natural language applications. We presented two novel algorithms grouping algorithm and chaining algorithm for clustering sets of documents, and which can handle a large set of documents and clusters. The two algorithms use semantic similarity and identical word measure, and their performance is much better than the performance of the K-means algorithm and the performance of the EM algorithm, used as a baseline for our evaluation.

Evaluating the system quality has been always a difficult issue. We presented an evaluation methodology to assess how the system clusters are related to the manually generated clusters using precision and recall measures.

The grouping and the chaining algorithm may be used in several natural language processing applications requiring clustering tasks such as summarizing set of documents relating the same event.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) research grant.

References

1. Manning, C.D., Schutze, H.: Foundations of Statistical Natural Language Processing. MIT Press (2000)
2. Berkhin, P.: Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA (2002)
3. Duda, R., Hart, P.: Pattern Classification and Scene Analysis. John Wiley & Sons, New York, NY (1973)
4. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: Five papers on wordnet. CSL Report 43, Cognitive Science Laboratory, Princeton University (1993)
5. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley Series in Computer Sciences (1989)
6. Galley, M., McKeown, K.: Improving word sense disambiguation in lexical chaining. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico. (2003)
7. Barzilay, R., Elhadad, M.: Using lexical chains for text summarization. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th European Chapter Meeting of the Association for Computational Linguistics, Workshop on Intelligent Scalable Text Summarization, Madrid (1997) 10-17
8. Silber, H.G., McCoy, K.F.: Efficiently computed lexical chains as an intermediate representation for automatic text summarization. Computational Linguistics **28** (2002) 487–496
9. Pantel, P., Lin, D.: Document clustering with committees. In: Proceedings of the ACM SIGIR'02, Finland (2002)
10. Over, P., ed.: Proceedings of the Document Understanding Conference, NIST (2003)
11. Over, P., ed.: Proceedings of the Document Understanding Conference, NIST (2004)