ALGORITHM FOR MAXIMAL FREQUENT SEQUENCES IN DOCUMENT CLUSTERING

László Kovács[1] – Helena Ahonen Myka[2]

1: University of Miskolc, Department of Information Technology,
kovacs@iit.uni-miskolc.hu

2: University of Helsinki, Department of Computer Science,
helena.ahonen-myka@cs.helsinki.fi

Abstract: In this paper, an efficient algorithm for maximal frequent sequences is described with an application in the document clustering processes. First, an overview on general document clustering process is given including the different feature reduction methods. Next, the role of word sequences and the methods for discovering the maximal frequent sequences are presented. In the last part, the document clustering based on oncept lattices with sequences is descriped.

1 Introduction

The most important purpose of the document or text database is to provide an efficient and flexible query system for the non-technical users. The query operation usually includes some kind of filtering or selection operation. The selection criteria are usually given by a list of keywords and a list of matching documents is returned. In the naive query evaluation method every document is tested in a sequential way for matching. The documents containing the given keywords are inserted into the result list. The main drawback of this evaluation method is the low efficiency. Due to the large number of documents this processing is very costly and takes a long time. The purpose of the researches is to improve the efficiency of the query processing algorithms in text databases.

One of the most useful methods of cost reduction is the pre-clustering or pre-classification of the documents. Based on the created cluster index structure, the set of tested documents can be reduced drastically. Another benefit of this method is that the document cluster hierarchy can be applied directly for query processing. The user may navigate in the hierarchy performing an interactive query based on relevance feedback. Many proposals have been published in the literature related to this topic, but due to the complexity of the semantic based evaluation of the documents, the problem of efficient text clustering can not be regarded as a

closed, finished research area. There are may open questions to be solved in the near future. This paper is devoted to present an efficient algorithm for determining the frequent word sequences in documents. First, an overview is given about the main clustering and classification methods, then the importance of word sequences is described. In the next, an algorithm is given for determining the frequent word sequences.

2 Problem area of clustering

A cluster is a collection of objects that are similar to one other within the cluster and are dissimilar to the objects in other clusters. The text clustering is the problem of automatically grouping of free text documents. The groups are usually described by a set of keywords or phrases that described the common content of the documents in the group. In the literature, there are several methods for the clustering process. The best known methods are the partitioning, the hierarchical and the density-based methods. In the descriptions, the letter N denotes the number of objects and M denotes the number of attributes.

In the case of partitioning methods, the algorithm constructs K partitions or clusters of objects. Each object must belong to only one cluster. The algorithm starts with an initial clustering. It then uses an iterative relocation technique to improve the partitioning quality. The goodness of clustering is measured by using a distance function defined on a pair of objects. The cluster is usually represented either by the mean value of the objects (k-means algorithm) or by the object located near the centre (k-medoids algorithm). In the initial phase, K objects are selected as cluster centres. For each of the remaining objects, an object is assigned to the cluster with the minimal distance. After inserting a new element into the cluster, the mean value of the cluster is recomputed. The termination criteria is usually given by a threshold of total distance value.

In the case of hierarchical methods, the clusters are generated by a hierarchical decomposition of the object set. The bottom-up approach starts with each object forming a separate cluster. In the next steps, the clusters close to each other are merged into a single cluster in a successive way until a termination condition holds. Taking a top-down approach, the initial cluster includes all of the objects. In each successive iteration a cluster is split up into smaller clusters.

A density-based clustering method grows regions with sufficienct high density into clusters and can discover clusters of arbitrary shape. Initially it checks every object by counting the number of neighbourhood objects within a given radius. If the number of neighbourhood objects exceeds a given threshold the objects is insert into the list of cluster core objects. If a core objects lies within the

neighbourhood of some other cluster the two clusters are merged into a common new cluster.

The common property of these methods is that they have high efficiency only in the case of low dimensionality. The main problem of text clustering is that there is no natural and efficient feature representation for a large set of different text documents. A text document as a complex object may have several hundreds of dimensions to describe the content of the document. It is desirable to find a reduced description method to increase the processing efficiency in text mining tasks. An another problem in text clustering is the representation of the generated clusters. The usage of the centre feature vector for representation is not always the best solution as a human interpreter would prefer short terms to a long list of attributes. It would be better to find a short term for cluster description that matches the exact feature centre as close as possible.

3. Document representation in the clustering process

To perform a clustering process, the objects should have some kind of attributes to measure the distance or similarity among the objects. These attributes are usually called as features of the object. Most of the proposals in this field consider the document as a set of words. In these representations, each feature corresponds to a single word found in the document set. As a document set may contain several thousand of words, these results in a very high impracticable dimensionality. To reduce the document space dimensionality some word reduction methods are applied in the pre-processing phase. The most common method to reduce the number of different words is to eliminate the words with low information value. These words are called stop words. The stop words are collected into a dictionary or a list. Another way for reduction is based on the statistical properties of the words: the infrequent and the frequent words are filtered out from the original text. This reduction is based on the assumption that the words with low frequency are not a characteristic word for the document set. The weight of a word is measured by its frequency. The stemming algorithm is also used in text clustering to remove some pre or suffixes from the words in order to determine the common root of the words.

There are two very basic observations about this representation based on words: first, the dimension of the feature vector space is very high, and the second, a great deal of information is discarded in this way.

Regarding the first problem, different feature selection methods are available to reduce the dimensionality of the feature space. The best known methods for dimension reduction are among others the followings: stepwise elimination

algorithm, decision tree based algorithms, principal component analysis, wavelet transformation [9].

The second observation is based on the fact that the words alone do not always represent the true atomic units of meaning. A straightforward consequence of this observation is to use phrases beside the single words in the text clustering process. A phrase is a chain of words occurring often in the document. The usage of word sequences provides a lot of benefits as it can be easily retrieved computationally and on the other hand, also a human-readable description of the document can be generated from the representation.

According to Lewis [7] the phrase-based text representation may work with low efficiency due to the following problems:

- the large set of occurring phrases
- uneven distribution of the feature vector
- many redundant features
- lot of noise in the set of occurring phrases

Although the usage of phrases is not so efficient as the methods based on single word representation, the benefits in the information gain is very important factor to go on with the investigation of this problem area. The main difficulty in using the phrase-based interpretation is the huge potential increase in the number of dimensions. If the number of words is equal to N, the number of phrases containing k words is N^k. To reduce the computational cost it is an important to find efficient algorithms for the different phases of the clustering process.

In the next section, the problem area of finding the frequent word sequences, is detailed and an efficient algorithm is represented. To describe the basic concepts, some denotations are introduced. Let S be a set of documents and each document consists of a sequence of words.

Definition: If a sequence p is a subsequence of q and the number of elements in p is equal to n, then the p is called an *n-gram* in q.

Definition: A sequence $p = a_1..a_k$ is a subsequence of a sequence q if all the items a_i occur in q and they occur in the same order as in p. If a sequence p is a subsequence of a sequence q we say that p occurs in q.

Definition: A sequence p is frequent in S if p is a subsequence of at least α documents in S where α is a given frequency threshold.

Only one occurrence of sequence in the document is counted. Several occurrences within one document do not make the sequence more frequent.

Definition: A sequence p is a maximal frequent sequence in S if there does not exists any sequence p' in S such that p is a subsequence of p' and p' is frequent in S.

One reason to extract maximal sequences, instead of fixed size frequent sequences, is that maximal sequences are both flexible and compact representation. Maximal sequences are flexible, since gaps are allowed in the sequences between words, which is important due to the many variations in the real texts. Maximal sequences also reduce overlapping information in the representation. Maximal frequent sequences can be used as content descriptor for documents. In this way, a document is represented by a set of sequences. This set can be used to discover other regularities in the document collection. As the sequences are frequent, their combination of words is not accidental and a phrase has a form that is present in many documents, giving a possibility to do similarity mappings for information retrieval or clustering.

In the literature, very few papers have been presented so far dealing with the problem of efficient algorithm for generating maximal frequent sequences. Related research includes discovery of frequent sets [1] and discovery of sequential pattern. In the context of textual data, a frequent set consists of words that co-occur frequently in documents where the order of words and the number of relative occurrences is not significant. So one word may occurs then times and the other one only once within the same word set. In some approaches [2], not only the list of maximal frequent sequences are discovered but the frequent set of frequent maximal sequences. This new level of data structure enables to determine the co-relation between the sequences. Using a co-relation matrix, the different frequent sequences may be clustered to better describe the semantic equivalence among the word sequences.

Most of the related approaches use a bottom-up processing. First, the frequent words are discovered, then the longer frequent sequences are iteratively formed from the shorter ones. In each iteration step, the usual why is to generate an initial set of candidate frequent sequences from the result set of the previous step. Next, all of the candidates are tested whether they are frequent or not. The problem with this approach is, that the number of possible candidate sequences may be very huge as the length of sequences increases. In this case, both the candidate generation as the candidate testing will be a very time consuming process.

In our proposal, the candidate sequences will be generated using an algorithm where the number of candidates remains limited, and the total cost of maximal frequent sequence discovery is also low.

4. Algorithm for the FMFS problem

During the development of the new algorithm, our intent was to find a low-cost method with a simple data structure. As the concatenation of two appropriate n-grams is the simplest way to find a candidate for frequent (n+1)-grams, this way

of candidate generation was selected. The rule for concatenation can be given in the following way:

If $a_1..a_n$ is a frequent n-gram on positions $p_1 \ldots p_n$ and $b_1..b_n$ is another frequent n-gram on positions $q_1 \ldots q_n$ where $a_2=b_1,a_3=b_2,..,a_n=b_{n-1}$ and $p_2=q_1,p_3=q_2,..,p_n=q_{n-1}$ then $a_1..a_nb_n$ is a candidate for frequent n-gram on position $p_1..p_nq_n$.

Using this principle for frequent n-gram generation, the corresponding data structure should store both the elements of the frequent n-grams as the positions of the n-grams. The simplest structure for this purpose is a list containing the frequent n-grams and their positions ordered by the starting position. Based on this list structure, the algorithm processes the sequences in a simple sequential way and candidates for frequent (n+1)-sequences are detected according to the given concatenation rule. For every document a list of candidate frequent (n+1)-grams is generated. In the next phase this list is scanned to determine the actual frequency value for every candidate. The candidates with a lower frequency value than a given threshold are removed from the candidate list.

Using this simple algorithm, the resulting algorithm does not provide yet the desired efficiency improvement. The reason for the high cost is that the number of candidate n-grams remained at a high level. To achieve a better efficiency the number of candidates should be reduced in additional processing steps. In our proposal, this reduction is based on the following elements:

1. The algorithm starts unlike to other systems not with the discovery of frequent 2-grams but with the generation of frequent 1-grams. This modification is based on the experience that most of the words are not frequent.

2. In each phases of the frequent n-gram generation some candidates may be eliminated if they are redundant candidates. A candidate is called redundant if all of the frequent grams of the next level can be generated from the reduced candidate set without the redundant candidates.

The detection of a redundant candidate is performed on the following way for the case of 2-grams. If $a_1 = (a_{11},a_{12})$, $a_2 = (a_{21},a_{22})$, $a_3 = (a_{31},a_{32})$ are all frequent and $a_{11} = a_{31}$, $a_{12} = a_{21}$, $a_{22}=a_{32}$ and $a_{11} = a_{31}$, $a_{12} = a_{21}$, $a_{22}=a_{32}$ and $a = (a_{11},a_{21},a_{22})$ is also frequent at this position then (a_{31},a_{32}) is a redundant 2-gram and so it can be eliminated from the candidate list.

According to this candidate elimination rule every frequent n-gram will be represented with a list of 2-grams. This list contains (n-1) 2-grams which are chained together:

$$(a_1,a_2), (a_2,a_3), (a_2,a_3), \ldots, (a_{n-1},a_n)$$

The end-point of any 2-gram is equal to the start-point of the next 2-gram. In general, the 2-grams that is a subset of union of some other 2-grams, are redundant 2-grams.

At the end of phase for the n-level, every n-grams that are not contained in any frequent (n+1)-gram is a maximal frequent n-gram and are inserted in the result list.

The simplified algorithm for discovering the maximal frequent sequences consists of the following steps:

1. Preparation phase
   - 1.1 Converting every document into a list of words with positions
   - 1.2 Reducing the set of words with the following methods:
     - Stop word elimination using a stop word dictionary
     - Determining the frequency of the words
     - Removing the words having too low frequency
     - The words are replaced by a numeric code
     - Building a word and code dictionary
   - 1.3 Saving the reduced decoded list of words into a temporary file
2. Discovery phase
   - 2.1 Discovering the set of frequent 2-grams
     - Creating a test window for scanning the temporary file
     - For every position of the window generating the 2-grams
     - Inserting the 2-gram candidates with the starting position into an intermediate list
     - Counting the frequency of the candidates in a separate loop
     - Saving the frequent 2-grams into a temporary file
   - 2.2 Discovering the set of frequent n-grams
     - For every level in increasing order
     - Scanning the condensed document list with the test window
     - For every position of the window generating the (n+1)-grams
     - Inserting the n-gram candidates with the starting position into an intermediate list
     - Counting the frequency of the candidates in a separate loop
     - Saving the frequent (n+1)-grams into a temporary file
     - Saving the not contained n-grams into the result set

- Validating the result set

The new elements of this algorithm can be summarised as follows. First, the filtering of frequent n-grams starts at level 1 and not at level 2 . Second, the generation of new candidates is performed on an efficient way to reduce the number of redundant candidate tests.

5. Implementation experiments

Although the algorithm allows sequences of any length, to make the discovery more efficient, we restricted the maximal distance of two consecutive items in the in a sequence. The maximal gap used in the tests is set to 2, i.e. if there are more than 2 words between two words in the document, they can not be used as neighbouring words in a sequence. This principle reduces the amount of pairs in the initial phase, which would otherwise need a lot of space and time.

The major data structures used in the implementation include a

- hash table that stores candidate n-grams with the exact occurrence positions
- hash table that stores the n-grams and the frequency values
- array (file) of candidate n-grams ordered by its position
- array (file) of maximal frequent n-grams

We have implemented the maximal sequence discovery algorithm in Perl. For experiments we have used a document set of moderate size. The number of documents was less than 500. In the next phase, the algorithm will be tested on the Reuters-21578 new collection which contains about 19000 documents. The average length of the documents is about 150 words.

During the tests, the algorithm was compared with an earlier program version for the maximal frequent sequence discovery. According to the test results, the modified algorithm can discover the required set of maximal frequent sequences and it provides an increased efficiency for smaller sets of documents. A more completed comparison can be given only after the tests with the Reuters data collection.

The discovery of frequent word sequences may be used as an important module in a larger document management system. The result of this module can be used among others for clustering purposes. Using phrases instead of single words, the users can better understand the semantic of the generated clusters.

The extraction of maximal frequent word sequences is a first step for further investigations to provide a better and semantic-full description of the documents

and of the document clusters. Among the possible research directions we can mention the involving grammar analysis of the frequent word sequences. In [3], we can find an approach to address this problem. It aims at discovering patterns which preserve as many features as possible such that the frequency of pattern still exceeds the frequency threshold given.

A common type of language analysis is to create a concordance for some word, i.e., to list all the sequences with the occurrences of the given word. This collection can provide useful information to gather some generalised knowledge about the use of the word. Based on this collection, the co-occurrences of some group of words can be discovered. These groups can be considered as clusters in the topic area.

6. Application of FMFS Algorithm in Document Clustering based on Concept Lattices

Concept lattices are used in many application areas to represent conceptual hierarchies stored in a hidden form in the underlying data. The field of Formal Concept Analysis [6] introduced in the early 80ies has grown to a powerful theory for data analysis, information retrieval and knowledge discovery.

The building of concept lattice consists of two usually distinct phases. In the first phase the set of concepts is generated. The lattice is built in the second phase from the generated set. We can find proposals in the literature for both variants, i.e. there are proposals addressing only one of the two phases and there are methods for combining these phases into a single algorithm. Based on the analysis of these methods, the cost for both steps is about the same order of magnitude and the asymptotic cost depends on mainly three parameters: the number of objects, the number of attributes and the number of concepts. The cost is always larger than the product of these parameters. The concept-set generation algorithms have two main variants. The methods of the first group work in batch mode, assuming that every element of the context table is already present. The most widely known member of this group is the Ganter's next closure method. The other group of proposals uses an incremental building mode. In this case, the concept set is updated with new elements if the context is extended with a new object. The Godin's method belongs to this group. Regarding the phase for building the lattice, the proposed approaches are based on the considerations that the lattice should be built up in a top-down (or bottom-up) manner because in this case only the elements of the upper (or lower) neighbourhood are to be localised. The second usual optimisation step is to reduce the set of lattice element tested during the localisation of the nearest upper or lower neighbour elements.

In the document clustering process based on concept lattices, the documents are considered as the objects[5]. The context is built up usually from the attributes of the documents. The words and the sequences stored in a document are taken as attributes of a document, so

$G = \{d_1,..,d_N\}$
$M = \{w_1,..,w_M\}$

Based on this interpretation, the context matrix can be built up on a simple way and the generation of the concept set and of the concept lattice can be performed using the methods presented in [5]. The only and large problem is the fact that the number of documents and the number of possible words and sequences is very huge. So the simple direct method can not be implemented within an acceptable execution cost value. This is the reason that an optimization step must be integrated into the basic algorithm. Analyzing the cost calculations, it is clear that the main cost factor is the number of the attributes *M*. The goal is to minimize the *M* value.

One of the simplest way to reduce the set of words used to describe the documents is to filter out the words based on a relevance value. As the result, only the words with higher importance remain in the documents. The relevance factor of a word emphases the words and sequences which occur only in few documents and the number of occurrences within a document is relative large. The relevance value can be given by:

$F_{ij} = c_{ij} * \log(N/N_j)$

where c_{ij} is the frequency of w_j in the document d_i. The symbol N_j denotes the number of documents containing the word w_j. Using this importance factor, only those words remain in the context which occur only in few documents. From the viewpoint of clustering, this is not always a desirable effect, as the different documents may contain different words and so there are no words occurring in several documents. The common words are needed to find out which documents belong to the same group, cluster. So it is useful to modify the original relevance factor by giving extra bonus points to the words common in a given number of documents. The optimal value of documents depends on the number of desired clusters. This value is an input parameter in our algorithm. An another performed modification is that instead of several local relevance value a global relevance value is calculated. Thus the applied importance factor is equal to

$F_j = \Sigma_i c_{ij} * \log(N/(1+(m-N_j))$

where *m* denotes the optimal number of clusters. Using this relevance value, the words and sequences contained in the document set can be ordered. After the filtering, only the words having the largest factor remain in the documents.

The concepts of the resulting concept lattice represent the sets of documents having common attributes. The number of nodes in the lattice is usually too large

to be presented it in whole to the users. It is reasonable to reduce the number of concepts to be used in the clustering process. The reduction is performed in the following way. If there are *N* objects in the context and the number of desired clusters is also *N*, then every object represents a cluster. On the other hand, if the number of desired clusters is smaller than the number of objects then more than one object will be mapped to the same cluster. In our proposal, the generated concepts are used as cluster centers. The objects are mapped to the closest concept from the selected ones. The distance between two attribute sets is defined as

$$d(c_1,c_2) = | \{a \mid (c1Ia \wedge \neg c2Ia) \vee (\neg c1Ia \wedge c2Ia)\} |$$

For a given *K* value, we can generate the set of concepts for which

$$\Sigma_{i,j} d(o_i,c_j) \Rightarrow \min,\ cj \in S\ ,\ |S| = K$$

holds and *i* denotes the document index, o_i the attribute set of the document, c_i is a concept from the set of cluster concepts *S*. The *S* set can be generated on different way, for example by using a gradient method. This method provides an approximate solution in an efficient way.

Based on the resulting clustering concept lattice, the query can be performed using a relevancy feedback method. This process can be divided into the following steps:

- providing the initial set of attributes user is interested in
- locating the cluster concept closest to the input attribute set
- returning the description of the selected cluster concept and the neighborhood of this concept
- the user responses determining which direction should be selected to refine the query

The structure of the proposed system is given in the Fig. 1.

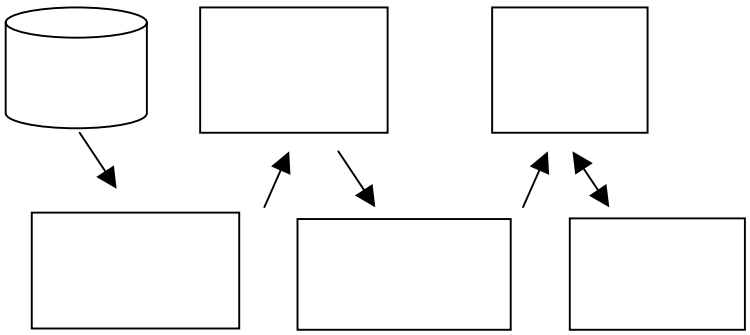

Fig. 1, Structure of Document Clustering System

The testing of the proposed query system is in progress. We have performed tests only with document set of smaller size until now. The program modules are implemented in Perl.

Acknowledgement

This work is partially supported by KFFP 180/2001

7. References

1. R. Agrawal – R. Srikant: Mining sequential patterns, International Conference on Data Engineering , March 1995
2. H. Ahonen: Finding all maximal frequent sequences in text, ICML99 Workshop, Machine Learning in Text Data Analysis, Bled, Slovenia, 1999
3. H. Ahonen-O. Heinonen – M. Klemmettien – A. Verkamo: Findig Co-occuring Text Phrases by Combining Sequence and Frequent Set Discovery, Proc. Of 16th International Joint Conference on Artificial Intelligence IJCAI-99, Stockholm, 1999
4. H. Ahonen: Discovery of Frequent Word Sequences in Text, The ESF Exploratory Workshop, London, 2001
5. L. Kovacs – P Baranyi: Document Clustering Using Concept Set Representation, INES02, Opatija, Croatia, 2002
6. B. Ganter – R. Wille: Formal Concept Analysis, Mathematical Foundations, Springer Verlag, 1999
7. D. Lewis-R. Schaipe-J. callan – R. Papka: Training Algorithms for linear text classifiers, Proceedings of SIGIR'96, 1996
8. Y. Yang – J. Pedersen: A Comparative Study of Feature Selection in Text Categorisation
9. J. Han – M. Kamber: Data Mining: Concepts and Techniques, Morgan Kaufmann, 1999