A Fast, Feature-based Cluster Algorithm for Information Retrieval

Martin Mehlitz
Technical University Berlin
DAI-Lab, Berlin, Germany
martin.mehlitz@dai-labor.de

Christian Bauckhage
Deutsche Telekom Laboratories
Berlin, Germany
christian.bauckhage@telekom.de

Şahin Albayrak
Technical University Berlin
DAI-Lab, Berlin, Germany
sahin.albayrak@dai-labor.de

Abstract

The Internet is a vast resource of information. Unfortunately, finding and accessing this information is often a very cumbersome task even with existing information platforms. Searching on the WWW suffers from the fact that almost every word is ambiguous to a certain degree in the information-rich environment of the Internet. Clustering search results is a way to solve this problem. This paper introduces a novel, fast way to cluster documents based on frequent term sets.

1 Introduction

With the Internet being available to more and more people and with the rapidly growing number of web pages, it has become a vast resource of information. Millions of people are searching the Internet for information every day. Even with well known search engines like Google or MSN, it may not be easy to find the information one is interested in. Clustering was introduced as a technique to improve Information Retrieval (IR) and the Cluster Hypothesis was put forth, stating that relevant documents tend to form clusters [11, 14]. In this paper we will introduce a new, fast method for clustering documents for IR and compare this approach to other cluster approaches.

After discussing related work we will briefly explain the notion of frequent sets in clustering and summarize common evaluation approaches in IR. Then, we introduce our novel method for document clustering. We also present its use in a prototypical search engine implementation and discuss its favourable computational complexity. Finally, we present results of an evaluation of our method obtained on a collection of news articles. A conclusion and an outlook to future work will end this contribution.

2 Related Work

Clustering is the process of grouping items such that items within a group are similar to each other and, simultaneously, dissimilar to the items in other groups [10].

Text clustering for information retrieval can be done either statically on the whole collection of documents [6, 11] or in a query-specific manner for a small number of top ranking documents returned by an inverted file search [5, 19]. Experiments regarding the effectiveness of text clustering for IR [5, 17, 19] concluded that the latter approach has the potential of improving IR.

Usually, novel document clustering methods for IR are compared to existing methods or to an inverted file search, based on recall and precision measures or some variants thereof [3]. A common evaluation methodology for such a comparison is the “Optimal Cluster Search” [9, 11, 15, 17, 19]. This methodology evaluates every single cluster and then selects the cluster with the optimal value as a representative for the cluster hierarchy.

Various techniques for clustering have been proposed. Examples include non-negative matrix factorisation [18], singular value decomposition [4, 7], suffix tree clustering [19], or formal concept analysis [5, 13]. All these techniques can be used to cluster search results in order to improve IR, but usually they do not scale well with an increasing number of documents. Therefore, we introduce an algorithm whose complexity does not depend on the number of regarded documents.

3 Background

Market basket analysis tries to find valid statements of the form “People who buy item A are likely to by item B”. A tool to reach such a statement are frequent item sets. The item set \( s = \{ A, B \} \) is considered a frequent set, if a rule of the form \( A \Rightarrow B \) has sufficiently high support and confidence values, with support and confidence defined as:

\[
\text{support} (A \Rightarrow B) = P(A \cup B) \quad (1)
\]
and

\[ \text{confidence} (A \Rightarrow B) = \frac{P(A|B)}{P(B)} \]

with respect to the collection of all regarded transactions. So, support is the portion of transactions that contain both items and confidence is the portion of transactions that contain A, given that they contain B. To further limit the number of sets, we will require in order for the set \( s = \{A, B\} \) to be frequent, that both \( A \Rightarrow B \) and \( B \Rightarrow A \) have sufficiently high confidence values. We use the Apriori algorithm \[1\] to mine frequent sets of terms from a collection of documents.

The measures typically employed to evaluate an IR system are precision and recall. Precision (“How many of the retrieved results are relevant?”) and recall (“How many of the relevant results are retrieved?”) are defined as:

\[ P(g, r) = \frac{g \cap r}{r} \]

and

\[ R(g, r) = \frac{g \cap r}{g} \]

where \( r \) denotes the number of retrieved documents, \( g \) corresponds to the number of relevant documents in the collection and \( g \cap r \) denotes the number of relevant documents among the returned ones. The effectiveness measure \( E \) is proposed in [11] as a measure for the evaluation of information retrieval systems that are based on text clustering. Effectiveness is defined as:

\[ E(P, R) = 1 - \frac{(\beta^2 + 1) \cdot PR}{\beta^2 \cdot P + R} \]

where \( \beta \) is a factor that determines the relative importance of precision and recall.

Frequent Term Clustering (FTC) and Hierarchical-FTC (HFTC) were introduced in [2]. Since they are related to our method, they will be compared to it in section 7. FTC is based on frequent item sets. A cluster consists of a frequent term set and all the documents supporting this set. Clusters are generated iteratively by selecting one frequent term set at a time, based on an appropriate scoring function. This function computes the overlap of each cluster with all other clusters. FTC keeps producing clusters until all documents from the document collection are present in at least one cluster. This process is shown in algorithm 3.1. HTCF is an extension of FTC, which runs several passes of algorithm 3.1 to generate a hierarchy of cluster. In the first pass, only frequent sets of cardinality one are regarded, building the first level of the hierarchy. Then, for each cluster, the documents are further clustered by regarding sets of cardinality two for the second level of the hierarchy and so on.

4 Class Item Set Mining

Steinbach et al [16] analyse characteristics of document classes, stating that “[...]each class typically has a “core” vocabulary of words that are used more frequently.[...] These core vocabularies may overlap, documents may use more than one “core” vocabulary, and any document may contain words from these different “core” vocabularies, even if it does not belong to the class of documents that typically uses such words”. This statement sums up the motivation behind many text clustering approaches: Identify the core vocabularies in a document collection to find document clusters. We follow this notion with a novel, fast cluster approach.

The approach that we propose works like this: First, we order the frequent sets by confidence and select a large number of frequent sets with a high confidence to be candidates for core vocabularies. This number needs to be large enough to make sure that frequent sets from each class are present as candidates. The rest of the frequent sets are then used to expand the candidates given that there is overlap between the candidate and the other frequent sets. This will yield sets of words that can be regarded as the core vocabularies. Eventually, we will discard those candidates that are complete subsets of other candidates. Therefore, we do not need to know the actual number of classes, since candidates from the same class will be expanded to the same (or very similar) sets. Following a “tree” analogy we call these starting points the seeds and the sets that we use to expand them sprouts. In the following, the sets of words that we generate with our method will be referred to as class item sets and the process of creating these sets as class item set mining (CISM).

In some scenarios (especially in collections that show classes with many subclasses) the class item sets that are produced might have a large overlap. To overcome this drawback we added a post-processing step that groups to-
Algorithm Class Item Set Generation

Let $I$ - the input - be the set of sprouts and $h$ be the cardinality of the largest mined frequent item set. Then the output $\Omega$ of this algorithm are the class item sets.

Input: $I$: the set of sprouts
Output: $\Omega$: the class item sets
1. Set $\Omega$ to the set of seeds
2. for $k \in \{2, 3, \ldots, h\}$
3. for $o \in \Omega$
4. for $i \in I$ with $|i| = k$
5. if $(overlap(i, o) = k - 1)$
6. then put the item which is not in $o$ in a set $t$
7. add $t$ to $o$
8. erase Class Item Sets that have a superset
9. erase Class Item Sets that have a superset
10. return $\Omega$

Algorithm 4.1: Class Item Set Generation

gether class item sets that have a large overlap assuming that these sets represent subclasses of the same class.

In the following we will describe the two steps of class item set mining in detail.

Given the frequent item sets we can generate class item sets. Algorithm 4.1 summarizes this procedure. First, the collection of frequent item sets is divided into two separate sets by taking a certain percentage of these item sets with the highest confidence values as the seeds and the remaining item sets as the sprouts. Every element in the set of seeds is a possible candidate for a class item set. Starting with the sprouts of cardinality two, each seed (line 3) will be compared to each sprout (line 4). If they have big enough overlap then the part of the sprout that does not overlap with the seed will be put in a separate set $t$ (line 6). After a seed has been compared to all sprouts this set $t$ will be added to the sprout (line 7). For a sprout of cardinality $k$, the overlap to a seed is considered big enough if exactly $k-1$ items of the sprout also appear in the seed. After this process of adding words to the seeds sets is finished for the sprouts of one cardinality (one iteration of line 2), item sets that have a superset among the other item sets will be erased (line 8). Finally, we erase again sets which have a superset among the class item sets (line 9).

Some item sets which are produced by the algorithm in step 1 can have overlapping phrases. In order to clarify the meaning of word sets and in order to enable a user to easily grasp the meaning of the class item sets we decided to arrange the word sets in a two level hierarchy. Step 2.1 groups similar word sets together, thereby getting the first level in our word set hierarchy. Then in Step 2.2, for each of these groups, we extract the overlap of the word sets and these overlap-sets together with the original sets build the second level of the hierarchy. The similarity of two item sets is defined as follows:

Let $I_1$ and $I_2$ be two item sets with cardinality $n_1$ and $n_2$. Let further be $n_1$ greater or equal $n_2$. $I_1$ and $I_2$ are similar if:

$$overlap(I_1, I_2) \geq d \ast n_2$$

(6)

Where $d$ is a real value between zero and one.

Step 2.1: Algorithm 4.2 shows the pseudocode for the algorithm that groups together similar sets. The algorithm first searches for the two sets which show the biggest overlap among all the sets in $I$ (line 2) and checks if they are similar or not (line 5). If they are not, then all the remaining sets form a group of their own (lines 14-15). If they are similar then the two form a new group (line 6) and all the remaining sets in $I$ that show similarity with these two sets will be added to this group (lines 7-10). Then the remaining sets will be searched again for two similar sets that may form a new group (line 2). The algorithm stops when all sets are assigned to a group.

Step 2.2: Once these groups are derived, for each group we extract the overlap from the item sets as shown in Algorithm 4.3. First, the item that occurs most frequently in the sets of the group is taken (line 2). Now, from all sets in the group that also contain this item (line 3) we compute the overlap (items that also occur in all of these sets) to generate a new item set (line 4). This overlap-set is added to the output group (line 5) as a new class item set. Then from the original sets in the group these overlapping items will be erased (line 6) as well as any empty item set that might result from this step (line 7). This process is then repeated.

Algorithm Class Item Set Grouping

Let $I$ - the input - be the set of all the created class item sets. Then the output $\Omega$ of this algorithm is a grouping of these class item sets.

Input: $I$: set of all the created class item sets
Output: $\Omega$: grouping of the class item sets
1. while ($I \neq \emptyset$)
2. set $i_1, i_2 = \text{biggestOverlappingSets}(I)$
3. $I$.delete($i_1$);
4. $I$.delete($i_2$);
5. if similar($i_1, i_2$)
6. then $g = \text{new Group}(i_1, i_2)$;
7. while (hasSimilarSets($g, I$))
8. $i_3 = \text{biggestOverlappingSet}(g, I)$;
9. $I$.delete($i_3$);
10. $g$.add($i_3$);
11. $\Omega$.add($g$);
12. else
13. for $i \in I$.add(new Group($i$));
14. return $\Omega$

Algorithm 4.2: Class Item Set Grouping
Algorithm Building Sub-Classes

Let $I$ be the input - be a group of class item sets created by the item set grouping algorithm. Then the output $\Omega$ of this algorithm is a group of item sets that are free of overlap.

Input: $I$: a group of class item sets created by the item set grouping algorithm

Output: $\Omega$: a group of item sets that are free of overlap

1. while (cardinality($I$) > 0)  
2. item $i$ = mostOccurringItem($I$);  
3. group tempGroup = $I$.getSetsThatIncludeItem($i$);  
4. Set $s$ = tempGroup.extractOverlap();  
5. $\Omega$.add($s$);  
6. $I$.deleteItemsFromAllSets($s$);  
7. $I$.deleteEmptySets();  
8. return $\Omega$

Algorithm 4.3: Building Sub-Classes

until the original group contains no more item sets.

Associated as a ranked list to each of this class sets are all the documents that contain the phrases in the set. The initial retrieval order is kept for the documents in each cluster.

5 Implementation

A prototype of an IR System that employs our cluster approach has been implemented [12]. Figure 1 shows the class item sets that have been mined for the search string “coca cola” from the first 200 pages returned from a search engine in April 2005.

Even though the search string “coca cola” is not ambiguous since it refers to a unique trademark, the results which are delivered by the search engine are still from different areas. This naturally happens since the phrase appears in different contexts. Also, there is no way to circumvent delivering pages from different contexts because given only this short phrase it is impossible to exactly infer what information was sought for. In this case, the cluster process can help by revealing some of the occurring classes so that a user can access the pages which the desired information.

In this example, the class item sets very accurately reflect different concepts that where (at the time of mining) important for the given search string. Concepts 3 and 4 represent the affairs of allegedly killed union workers in Colombia and of an allegedly caused water shortage in the Kerala region. Concept 3 refers to pages that offer collectible items of the Coca Cola Company. Further identified classes include aspects like “advertising” (1.2) or “history” (1.6).

Figure 1. Example of Class Item Set Mining

6 Complexity

The complexity of the algorithms FTC and HFTC depends on the number of frequent sets $f$ and the number of documents $d$. The complexity of the CISM algorithm, in contrast, depends only on the number of frequent sets. However, the number of frequent sets that are mined for a collection of documents is almost constant given a fixed minimum support value for frequent sets.

For the CISM algorithm, comparing each frequent set in the seed set with each frequent set in the sprout group, yields a worst case complexity of $\frac{1}{2}f \times \frac{1}{2}f = \frac{1}{4}f^2$. The later steps for grouping and overlap extraction require, in the worst case (every seed is cluster), the same number of comparisons. Assuming $f$ to be constant, this yields a complexity of $O(1)$.

The worst case scenario for FTC would be a scenario where every frequent set is contained in every document. Computing the clusters would then compute for each frequent set the overlap of each document with each other frequent set, yielding a complexity of $O(df^2)$, which, considering $f$ a constant, reduces to $O(d)$.

The complexity of the HFTC scenario is the same as the FTC complexity. Even though the number of comparisons of frequent sets is smaller since every pass only compares sets of the same cardinality, this is only a constant factor which does not change the complexity of the algorithm.

7 Experiments

The CISM algorithm has been evaluated and compared to the FTC and HFTC algorithm. In this section the results
Experiments were run on a computer with a Opteron 2.4 GHz processor with 2 GB of RAM under a Linux operating system. All algorithms have been implemented in Java, using the same classes wherever possible.

Since an evaluation of Web search results is very subjective and since it is impossible to reproduce these results, because the WWW is constantly changing, we chose to evaluate our cluster approach on the collection of approximately 170,000 Wall Street Journal documents of the Tipster corpus. We use the TREC1, TREC2, TREC3 and TREC4 topics of the Ad Hoc Track to evaluate our algorithm. Each of these 200 topics represents a certain information need. Given for each such topic is a list of relevant articles.

The optimal cluster search evaluation methodology is employed to measure quality of a cluster structure. It evaluates each single cluster and then selects the cluster which yields the optimal value for the employed measure. Since we are offering our result sets directly to a human user, we can assume that the precision is the most important aspect for measuring the retrieval performance. A human user will not be interested to know what fraction of the relevant documents is retrieved as long as (s)he sees enough relevant documents. Since we regard the first fifty documents that are returned by each IR system we compute and compare the values for precision among the top 5, top 10, top 20 and top 50 documents.

For each topic the topic’s title (or the topic’s description if no title is given) is used to query a Lucene index over all the WSJ data. Then the 1000 top ranking documents are preprocessed and the 35 terms with the highest term-frequency/inverse document frequency (TFIDF) values are extracted for each document. Frequent term sets are then mined from these term sets. The frequent sets are then forwarded to the cluster algorithms.

All three considered algorithms have free parameters that influence the generated cluster structure. Therefore, each algorithm first goes through a short training phase based on a grid selection method. A training set was constructed from the 200 TREC topics by randomly selecting 160 topics (40 from each TREC). Then, for several specific combinations of values for each algorithms parameters, the average precision values are computed for the training set and the best models are then selected for evaluation.

We measured the performance of the trained versions of the CISM, FTC and HFTC algorithm on the remaining 40 topics. Table 1 shows the statistics for the initial retrieval of documents. Table 2 displays the precision values at several cut-off values achieved by the inverted file search (IFS) as well as by each cluster method, while Table 3 shows the values of the effectiveness measure for the cluster algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>@5</th>
<th>@10</th>
<th>@20</th>
<th>@50</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFS</td>
<td>0.260</td>
<td>0.233</td>
<td>0.201</td>
<td>0.17</td>
</tr>
<tr>
<td>CISM</td>
<td>0.405</td>
<td>0.315</td>
<td>0.256</td>
<td>0.169</td>
</tr>
<tr>
<td>FTC</td>
<td>0.420</td>
<td>0.340</td>
<td>0.289</td>
<td>0.209</td>
</tr>
<tr>
<td>HFTC</td>
<td>0.475</td>
<td>0.400</td>
<td>0.309</td>
<td>0.191</td>
</tr>
</tbody>
</table>

All cluster methods manage to improve the document retrieval. Although, CISM shows slightly worse retrieval performance than FTC and HFTC, CISM clearly outperforms the other algorithms in terms of clustering time.

Using the same test set of 40 topics we evaluated the clustering time in regard to the number of documents that are returned by the initial retrieval; averages are given in Fig. 2. Since for all algorithms the clustering time depends on the number of features from which the cluster structures are generated, we also show in Fig 3 the number of features from which the algorithms build clusters.

### Table 1. Initial Retrieval

<table>
<thead>
<tr>
<th>Average Number Retrieved</th>
<th>Average Number Relevant retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>982,525</td>
<td>37.2</td>
</tr>
</tbody>
</table>

### Table 2. Precision Values at Different Cut-offs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>@5</th>
<th>@10</th>
<th>@20</th>
<th>@50</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFS</td>
<td>0.260</td>
<td>0.233</td>
<td>0.201</td>
<td>0.17</td>
</tr>
<tr>
<td>CISM</td>
<td>0.405</td>
<td>0.315</td>
<td>0.256</td>
<td>0.169</td>
</tr>
<tr>
<td>FTC</td>
<td>0.420</td>
<td>0.340</td>
<td>0.289</td>
<td>0.209</td>
</tr>
<tr>
<td>HFTC</td>
<td>0.475</td>
<td>0.400</td>
<td>0.309</td>
<td>0.191</td>
</tr>
</tbody>
</table>

### Figure 2. Time required for Clustering

8 Conclusions and Future Work

This paper introduced a novel, fast way to cluster documents based on frequent term sets. This work was motivated by a common observation in the context of web searches. Finding and accessing this information is often a very cumbersome task, since searching the web suffers from the fact...
Table 3. Cluster Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Effectiveness</th>
<th>Average Cluster Size</th>
<th>Average Number of Clusters</th>
<th>Average Time for Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>CISM</td>
<td>β = 0.5 0.85</td>
<td>131.69</td>
<td>22.825</td>
<td>415 ms</td>
</tr>
<tr>
<td></td>
<td>β = 1.0 0.826</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>β = 2.0 0.768</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTC</td>
<td>0.809</td>
<td>32.975</td>
<td>85.199</td>
<td>8994 ms</td>
</tr>
<tr>
<td>HFTC</td>
<td>0.751</td>
<td>40.049</td>
<td>243.175</td>
<td>11944 ms</td>
</tr>
<tr>
<td></td>
<td>0.756</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.713</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Regarded Features

that almost every word is ambiguous. Clustering can help revealing class structures in the retrieved results. However, most clustering techniques are very time consuming and are thus of limited use for online applications where the user demands timely processing.

An empiric evaluation has shown that our CISM approach can, independent of the number of regarded documents, cluster documents in meaningful classes. While its performance compares to that of other approaches based on frequent term sets, it clearly outperforms them in terms of computation time, simply because its computational complexity is of a smaller order of magnitude.

In future work we will examine whether the use of features other than TFIDF weighted terms can further improve the retrieval behaviour.

References