Personalized Hierarchical Clustering

Korinna Bade, Andreas Nürnberger
Faculty of Computer Science,
Otto-von-Guericke-University Magdeburg, D-39106 Magdeburg, Germany
{kbade,nuernb}@iws.cs.uni-magdeburg.de

Abstract

A hierarchical structure can provide efficient access to information contained in a collection of documents. However, such a structure is not always available, e.g. for a set of documents a user has collected over time in a single folder or the results of a web search. We therefore investigate in this paper how we can obtain a hierarchical structure automatically, taking into account some background knowledge about the way a specific user would structure the collection. More specifically, we adapt a hierarchical agglomerative clustering algorithm to take into account user specific constraints on the clustering process. Such an algorithm could be applied, e.g., for user specific clustering of Web search results, where the user's constraints on the clustering process are given by a hierarchical folder or bookmark structure. Besides the discussion of the algorithm itself, we motivate application scenarios and present an evaluation of the proposed algorithm on benchmark data.

1. Introduction

Most people structure their data in order to better keep track of where certain information can be found. This can be seen on a small scope (e.g. personal files in a folder structure) as well as on a large scope (e.g. web directories or library catalogs). Furthermore, this structuring is usually hierarchical as this allows for differentiating between documents on different levels of granularity. Despite the usefulness of presenting documents in hierarchies, most of the documents available, e.g. through the Internet, are still provided without any categorical information that would allow to assign or present these documents in a hierarchical structure. This is mainly due to the facts that this structural information usually has to be created manually – which is a very time consuming and expensive task – and that many ways to structure collections are possible.

Besides, people have different ideas what a good structure is, mainly because of their different perspectives, knowledge and cultural background. So one might argue that creating a uniform hierarchical structure for all users on the available data can at best be partly beneficial for the individual users. More useful would be a structuring that obeys the user's intention of a good structure. In this paper, we therefore propose a method that is capable of structuring an unknown dataset by integrating some knowledge about the structuring behavior of the user.

As a scenario, consider a user's interaction with the Web over a search engine to locate some new valuable information. As a result, the user usually gets a huge list of documents. Even though this list is ranked by relevance, the relevance is determined by generally applicable criteria, which might not correspond to the user's search intention. This is frequently the case for ambiguous search terms or for very specific information needs that require several search terms to be clearly defined. Therefore, in order to determine whether the information searched for is part of the retrieved list, the user has to scan several documents. A clustering of the search results could decrease the number of documents that a user must actually look at to determine whether the information searched for is contained within the search results. Large amounts of data could be skipped as the user can ignore whole clusters based on the assumption that documents inside one cluster have similar content.

Clustering the data by a standard clustering method that considers only information about the documents itself will provide one possible structure. However, we want to find the most useful one for the current user. How can the system gain knowledge about the user preferences for this task? The user usually has created some kind of hierarchical structuring, for example, by his folder structure or Web pages he has already visited and stored in a (hierarchically) ordered collection of bookmarks. Of course, these hierarchies only cover a very small part of the document collection and its topics, but most likely a large part of the currently most interesting topics from the user's point of view. Information about this personal structure could therefore be integrated when looking for the best clustering for a specific user.

More formally, we can summarize this setting as a semi-supervised learning problem. There is some training data available, for which a hierarchical class structure is known. Furthermore, there is a lot of data without any class information. However, we still have information about the similarity between the data items. Please note that the available training data does not cover all existing classes. Instead, a large number of classes is also unknown. Therefore, classification methods cannot be applied for this task.

The approach presented in the following tries to find a hierarchical structuring of the data using a hierarchical agglomerative clustering approach, which integrates the known class structure available. The resulting dendrogram could then be utilized to derive some class knowledge for documents without class information or derive at least groups of similar documents that belong together, however, without a class label as this is not yet known.

Our paper is structured as follows. In the next section, we review some related work on semi-supervised clustering. In Sect. 3, we present our approach to personalized hierarchical clustering, which we evaluate using a benchmark dataset in Sect. 4. Finally, we give some conclusions and briefly discuss ideas for future work.

2. Related Work

The clustering methods presented in the following belong to the constraint based clustering methods, which are described in more detail in the following section. These algorithms always try to learn a flat clustering with constraints based on a flat class structure. In contrast to this work, we try to derive a hierarchical cluster structure with constraints based on a hierarchical user structure. To our knowledge, such an hierarchical approach was not presented before.

In [7], constraint information is used in two ways. First, a weight matrix describing term influence is learned according to the constraints. This matrix is then facilitated using the Mahalanobis distance during the clustering phase. Furthermore, a hierarchical agglomerative clustering algorithm is initialized with groups of documents formed by the constraints instead of using single documents. Cluster merges that violate constraints are not performed. The hierarchical algorithm is not used to derive a hierarchical structure, but is terminated after the desired number of flat clusters is found. A similar approach that also learns such a weight matrix is described in [14]. They use an optimization approach for learning the weights and use K-Means for clustering.

An algorithm, which uses constraints solely to modify the clustering process, is introduced in [13]. It is based on K-Means. Here, a document is not assigned to the closest cluster prototype but instead to the closest cluster prototype that does not violate any constraint. The authors of [1] alter K-Means in a similar way, however instead of prohibiting a cluster merge that violates any constraint, they introduce a cost function that penalizes constraint violations. Data is then assigned to the prototype with minimal costs, which minimizes the overall cluster costs to a local optimum. They also propose an active learning strategy to find the most suitable constraints. The authors of [5] also use constraints to compute a weight for the importance of a term in the distance measure. These constraints are learned by user feedback, which is similar to the active learning scenario of the previous approach. Their document model is based on probability distributions.

In [8], an approach is presented that also tries to change the underlying distance measure. However, they modify the elements of the distance matrix directly instead of adapting the describing function. The distance values of elements that should be merged together due to constraints are reduced to 0. Other elements in the matrix are then influenced by trying to preserve the triangle inequality property. We also used the idea of term weights in prior work [9]. Here, the weights were used to incorporated user feedback about group assignments in a self-organizing map.

3. Personalized Hierarchical Clustering

Our approach to personalized hierarchical clustering can be described as a semi-supervised setting. In a standard semi-supervised setting, a set of data items is given. For some of these items, class labels are known and for some they are not. However, usually the number of possible classes and all existing class labels are known a-priori. In contrast to this, in constraint based clustering, background knowledge is provided by pairwise constraints, expressing whether two items belong together or should be separated. Our setting considered here is different to both of these settings in two ways. First, we know some class labels but not all existing ones. Second, our constraints between items are defined by an underlying hierarchical structure instead of assuming a flat cluster structure. Therefore, we define our problem setting as follows.

Given is a set of classes $C = C_k \bigcup C_u$, out of which only the classes in C_k are known in advance. The elements in C_u are unknown as well as their number. Furthermore, there is a set of relations between classes $R_H = \{(c_1, c_2) \in C \times C | c_1 \geq_H c_2\}$, which describes the hierarchical relations between two classes ($c_1 \geq_H c_2$ means that c_1 contains c_2 as a subclass). The relations between the known classes are denoted by R_{Hk}, which is a subset of R_H. Thus, the combination between C and R_H represents a hierarchical tree structure of classes.

In addition, there is the set of documents $D = D_k \bigcup D_u$ that should be structured. For some documents, class information is available, given by the training set $T_k = \{(d, c) \in D_k \times C_k\}$, i.e. a class label is available for each instance

in D_k. Please also note that the notion of semi-supervised is therefore two-fold. First, not all classes are known in advance. Second, the training data has only partly class information assigned to it.

For the unlabeled documents, i.e. the documents in D_u, we are looking for the mapping to classes in C (i.e. $C_k \bigcup C_u$), denoted by $T_u = \{(d, c) \in D_u \times C\}$. This means that, first, the algorithm should derive mappings to known classes for documents in D_u, if such mappings exist. Second, it should find new classes, i.e. classes in C_u, by grouping similar documents and assigning a class label to this group. Most importantly, the relations of these new classes to the existing ones have to be found. It is an important feature of the algorithm to preserve the existing knowledge about class structure and document assignment.

In constraint based clustering, the supervised knowledge is expressed by two sets, the must-link constraints and the cannot-link constraints. Each set contains pairs of documents. The pairs in the must-link-set describe documents that should be clustered in the same cluster, and the cannot-link set contains document pairs that should be separated. In hierarchical clustering, each document is actually linked with every other document. What matters here is the order, in which the documents are linked. We therefore suggest an extension in form of a set describing must-link-before (MLB) constraints. An element of this set is a pair of a document $d \in D_k$ with an ordered list of m document sets S. Each of these sets contains documents from D_k, to which d should be linked before all documents in the following sets. The order between the sets is important, while inside the sets it does not matter, which document is linked first. Each MLB constraint must cover all documents in D_k. Therefore, application of the MLB constraints requires knowledge on all relations between the elements of D_k, which is, e.g., true, if background knowledge is given in form of a class hierarchy and labeled training data. The definition of the MLB constraint can be formalized as

$$MLB = \{(d, (S_1, ..., S_m))\} \quad (1)$$

with $D_k = \bigcup_i S_i \cup d$ and $\forall i, j (i \neq j) : S_i \cap S_j = \emptyset$ and $\forall i : d \notin S_i$.

To make things more clear, consider the hierarchy given in Fig. 1. Here, a document from class 4 should first be linked with all documents in 4. Then, it should be linked to documents from class 3 and 5, whereas the order does not matter. Finally, the document should be linked to all other documents, i.e documents from class 1 and 2. This leads to the MLB constraint as shown on the right side of Fig. 1.

The proposed clustering approach BiHAC (Biased Hierarchical Agglomerative Clustering) described in the following is derived from a hierarchical agglomerative clustering algorithm and utilizes the MLB constraints to supervise the clustering process. For the current implementation,

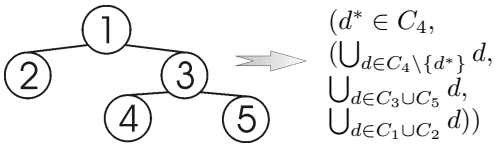

Figure 1. MLB Constraint Extraction (Ex.)

we assume centroid linkage for the computation of cluster similarities. However, the approach can also be used for other linkage types. The underlying similarity measure is changed to take term weights into account during clustering. These weights are learned by integrating the given constraints. In the next section, we motivate the usage of term weights and describe their influence in the clustering process. In Sect. 3.2, we then present how appropriate term weights can be learned, while in Sect. 3.3, we give some remarks on the termination of the algorithm.

3.1. Utilizing Term Weights

In order to to be able to compare the similarity of documents, we use the standard vector space representation as introduced in [11]. Thus, each document is represented by a numerical term vector, in which each element represents one term in the dictionary of all terms and the value represents the importance of a term within the document. We compute $tf \times idf$ document weights as introduced in [10] to describe each document.

However, using all terms of all documents leads to a very high dimensional data representation that is very difficult to cluster, as with increasing dimensionality the distances between objects become more similar (see [2]). To circumvent this problem, we can use term selection techniques to remove terms that are not statistically significant in differentiating documents from each other [6], and/or we can try to increase the 'importance' of terms that are more appropriate to distinguish documents from each other and decrease the 'importance' of the other terms. This can be done by introducing term weights in the measures used to compute the similarity between documents as described in the following. If we furthermore assume that important terms for distinguishing documents vary between users, we can use this idea to create user specific similarity measures by allowing the use of different weighting schemes for each user.

Summing up, highly weighted terms can be understood as expressing the importance of terms for the user's classification behavior. These terms therefore also describe the user's topics of interest. Such information could, e.g., be used in a user profile and further facilitated, e.g., for grouping users with similar interest. On the other hand, weighting information of terms already available, e.g., from a user profile or from manual user input could be applied to our algorithm. That means that known keywords could be directly

used in the clustering process or used as an initialization for further weight adaptation.

Thus, with term weights we project the user's notion about what terms are important into the clustering process. This basically distorts the similarity space to make documents more similar that are defined to be similar by the user and more dissimilar if the user has separated them.

Formally, we can combine all term weights in a vector $w = (w_1, \ldots, w_n)^T$ containing a weight for each term in the dictionary. Furthermore, it is required that each weight is greater than or equal to zero and that its sum is bound to the number of terms to avoid extreme case solutions:

$$\forall w_i : w_i \geq 0 \qquad \sum_i w_i = n \tag{2}$$

As mentioned above, the weights are used to influence the similarity measure used during clustering. In the following, we use the commonly used cosine similarity, i.e. the scalar product of two normalized document vectors [11]. The weighted cosine similarity between two documents d_1 and d_2 given a weight vector w is then computed by:

$$\mathrm{sim}(d_1, d_2, w) = \sum_i w_i \cdot d_{1,i} \cdot d_{2,i} \tag{3}$$

Note that the adapted cosine similarity as calculated by (3) is no longer bound to the interval $[0; 1]$. However, a larger value still represents more similar documents.

3.2. Learning Term Weights

The weights for a given setting are determined by an iterative approach. In each iteration, the data is clustered using a hierarchical agglomerative clustering algorithm that uses the weighted cosine similarity to determine document similarities. As an initialization, all weights are set to 1, which results in standard clustering. At the end of each iteration, the weights are updated. The algorithm stops, if the weights do not change any more or some other stopping criteria holds (see Sec. 3.3 for further details).

The weight adaptation is accumulated over one complete clustering process (batch learning), i.e. one iteration of the learning algorithm. This is necessary as changes in the weight vector change the document similarities and therefore the whole clustering process. Each time two clusters are merged as part of the hierarchical clustering algorithm, it is checked whether this merge causes a violation to the given constraints i.e. the given class structure of the user. In this case, a weight adaptation is computed to make these two clusters more dissimilar. This can be realized in multiple ways. To guide the weight adaptation, we want to make the two clusters also more similar to clusters, with which they should have been merged according to the user structure. For each of the two clusters, we therefore search a cluster, with which merging does not violate any MLB constraint. Out of all such clusters, we then select the one closest as this requires the fewest weight changes.

For each of the two wrongly merged clusters, the following setting applies. Given that cluster similarity is determined by centroid linkage, we consider the centroids of three clusters, the centroid of the cluster at hand c_c, the centroid of the cluster, to which c_c should become more dissimilar, c_d, and the centroid of the cluster, to which c_c should become more similar, c_s. Then each weight is adapted by:

$$w_i' = w_i + \eta \cdot (c_{c,i} \cdot c_{s,i} - c_{c,i} \cdot c_{d,i}), \tag{4}$$

where η is the learning rate and controls the strength of the weight adaptation. In our current implementation, we use a constant value, however it is also possible to use different strategies like simulated annealing here. We want to further investigate the influence of η in future work.

What effect has this weight adaptation with respect to the weighted cosine similarity? If a term contributes more to c_s than to c_d, the weight of this term is increased and otherwise decreased. The increase (decrease) of a single weight increases (decreases) each similarity value in general. However, if a term has a higher value in the document vector for a certain document in contrast to another, the increase (decrease) in the similarity measure is also stronger. Therefore, the similarity between c_c and c_s is increased more (decreased less) than the similarity between c_c and c_d if the term contributes more (less) to c_s than to c_d. Therefore, the weight adaptation actually makes c_c more similar to c_s and more dissimilar to c_d.

The question that remains is which cluster merges require a weight adaptation, i.e. identifying the merges that violate the constraints. In each iteration of the learning algorithm, we want to adapt the weights as much as possible as each clustering run is expensive. However, we also do not want to penalize a wrong cluster assignment more than once as this might lead to too strong adaptations for certain weights. Therefore, we have to consider the hierarchical nature of the clustering process. For these reasons, we applied the following rules to decide whether a weight adaptation is needed: If both clusters contain only items from one user class, no weight adaption is performed. Otherwise, the algorithm checks whether complete subclasses of one user class are merged. If this is not the case, the hierarchy is violated. However, as mentioned, another violation might have already been occurred in an earlier merge that we do not want to penalize again. This is true, if elements of a subclass are contained in both clusters. Hence, the weights are not adapted under these circumstances.

3.3. Termination of the Algorithm

The most natural stopping criterion for the weight adaptation would be a correct clustering of the known documents, i.e. a clustering that does not violate any constraints. However, such a solution might not exist. Therefore, we need another criterion. Most importantly, we need a measure to evaluate the quality of the current clustering to be able to decide whether the algorithm is still improving.

For this purpose, we define the cluster error CE. For each document d with a given constraint $(d, (S_1, ..., S_m)) = (d, \mathcal{S})$, we can derive from the dendrogram of the current clustering the order, in which documents are merged to d, producing another ordered list of document sets $\mathcal{R} = (R_1, ..., R_r)$. To compute CE, we count how often the order in $\mathcal{R}$ violates the order in $\mathcal{S}$, i.e. the order of two documents d_i and d_j is reversed. In addition, violations between classes that are further apart in the hierarchy are more severe. Therefore, constraint violations should be weighted. The distance between two classes is reflected in the distance between two constraint sets in the ordered list and can therefore be utilized for weighting. The overall cluster error CE can therefore be defined as:

$$CE = \sum_{(d,\mathcal{S},\mathcal{R})} \sum_{\substack{(d_{i_{k,x}}, d_{j_{l,y}}) \\ k<l}} \left\{ \begin{array}{ll} x-y & \text{if } x > y \\ 0 & \text{else} \end{array} \right\}, \quad (5)$$

where $d_{i_{k,x}} \in R_k$, $d_{i_{k,x}} \in S_x$, $d_{j_{l,y}} \in R_l$ and $d_{j_{l,y}} \in S_y$.

The cluster error can directly be used to define a stopping criterion for the algorithm. If the cluster error could not be decreased in a certain number of iterations, the algorithm is supposed to not find any better solution any more and stops. The complete algorithm for learning the weights is summarized in Fig. 2.

4. Evaluation

We evaluated BiHAC with a part of the banksearch dataset [12], consisting of 450 web pages in a two-level hierarchy. We used the classes and number of documents as shown in Fig. 3. After parsing the documents, we applied standard document preprocessing methods, e.g. filtered out terms that occurred in less than five documents, in almost all documents ($> |D| - 5$), stop words, terms with less than four characters, and terms containing numbers. Each document that had no terms left after these preprocessing steps was ignored. In our dataset, this was true for two documents, one from the Insurance Agencies class and one from the Sport class. Finally, we selected out of the remaining terms the best 4000 as described in [3]. The method uses an entropy measure and ensures an about equal coverage of the documents by the selected terms. Term selection was done to increase clustering speed.

weightAdapt(MLB, D)
- Initialize w: $\forall w_i : w_i := 1$
- Do:
  - Initialize clustering with D
  - $w^{(a)} := \vec{0}$
  - While not all clusters are merged:
    - Merge two closest clusters c_1 and c_2
    - If MLB is violated by merge:
      - Determine the most similar clusters $c_1^{(s)}$ and $c_2^{(s)}$ to c_1 and c_2, respectively, that can be merged according to MLB
      - $\forall i : w_i^{(a)} := w_i^{(a)} + (c_{1,i} \cdot c_{1,i}^{(s)} - c_{1,i} \cdot c_{2,i})$
      - $\forall i : w_i^{(a)} := w_i^{(a)} + (c_{2,i} \cdot c_{2,i}^{(s)} - c_{1,i} \cdot c_{2,i})$
  - Compute current cluster error CE
  - Adapt $w := w + \eta \cdot w^{(a)}$
  - $\forall i$: if $w_i < 0$ then $w_i := 0$
  - Normalize w such that $\sum_i w_i = n$ holds
- Until $w^{(a)} = \vec{0}$ or CE did not improve for r runs
- Return w

Figure 2. The weight adaptation of BiHAC

- **Finance** (0)
  - Commercial Banks (50)
  - Building Societies (50)
  - Insurance Agencies (50)
- **Sport** (50)
  - Soccer (50)
  - Motor Sport (50)
- **Programming** (0)
  - C/C++ (50)
  - Java (50)
  - Visual Basic (50)

Figure 3. Class Structure of our Training Data

4.1. Evaluation Procedure

It is difficult to find a good evaluation strategy to measure the quality of cluster personalization. Usually, user studies are necessary, however they are very time consuming and not in scope of this paper. Furthermore, we deal with a hierarchical class structure in the data. And third, our algorithm provides so far a very fine-grained structure by the dendrogram giving the complete nested dichotomy of the data. Although this could be displayed to the user, a more coarse structure would improve navigation. Therefore, we decided on a two-fold evaluation procedure. First, we measure how well the hierarchical structure of the data was learned. Second, we measure the classification quality that can be gained, independently of the hierarchical relations between the classes.

To compare the learned cluster structure with the given class structure, we used the MLB constraints in a similar way as for the cluster error presented in (5). However, for

evaluation purposes, the relations between all documents in D are defined by MLB constraints. Then we can use the cluster error CE as a measure of how good the user structure was learned. A value of 0 denotes a perfect match.

For measuring classification quality, we need to assign labels to unlabeled documents. A common strategy in semi-supervised clustering is to label unlabeled documents in the same cluster of a labeled item with this label. However, our algorithm does not actually extract distinct clusters. We therefore follow a simple strategy. From the dendrogram, we extract the largest possible clusters that only contain items with the same class label. For super-classes, we consider each document directly assigned to this class or assigned to a subclass as labeled with this class. Note that extracted super-clusters therefore contain sub-clusters, which is in correlation with the given class structure. A very simple example is given in Fig. 4. The underlined numbers are the labeled data items available.

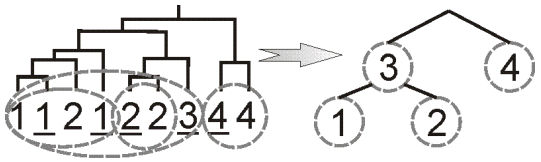

Figure 4. From Dendrogram to Classes (Ex.)

With this derived class structure, we evaluate the classification quality of the algorithm with a standard evaluation measure from classification, the F-score (as a combination of precision and recall). The precision of a class c is the fraction of all documents retrieved for this class that are correctly retrieved, while the recall of c is the fraction of all documents belonging to this class that are actually retrieved. The F-Score combines both measures to describe the found trade-off between the two (see e.g. [6]).

Please note that these measures are computed on the unlabeled data only as the labeled data is correctly labeled by default. This also allows us to compare our approach to a standard classifier as a baseline. Here, we trained a support vector machine (SVM; based on the libSVM implementation [4]). As mentioned above, the problem itself cannot be handled by a classifier in general. However, in this evaluation, we deal with a special case, where C_u is empty. Therefore, a classifier can be applied for comparison.

When computing precision and recall for higher level classes in the hierarchy, we decided to evaluate these classes as a whole, i.e. all elements from sub classes are handled as if they belong to this class. This helps to compare, how the performance changes on different hierarchy levels.

To evaluate BiHAC, we used different numbers of labeled documents per class: zero (which leads to the standard hierarchical agglomerative clustering), one, five, ten, and twenty. With these selected documents, we formed the MLB constraints used by the algorithm for learning the weights. Two results of each evaluation run were captured. One after a fixed number of five runs (named with BiHAC-5) and one with maximally minimized cluster error on the training data (i.e. the labeled data).

4.2. Results

Figure 5 shows the cluster error of the complete data. More specific, the percentage of violated constraints is presented. As can be seen, our approach is capable of adapting towards a user specific class structure. Already a single labeled document per class reduces the cluster error CE from 40% to 31%. More labeled documents can decrease the cluster error even further, although improvements get less because many important terms occur in several documents. This means a new document also provides information that is already known. Interestingly, large improvements can already be gained after five iterations of the algorithm. The more training data available, the more similar are the gained improvements in cluster error. Moreover note that the minimum of the cluster error is most likely not zero as there are conflicting constraints, either introduced by noise or due to the fact that with creating the document vectors valuable information about the documents got lost.

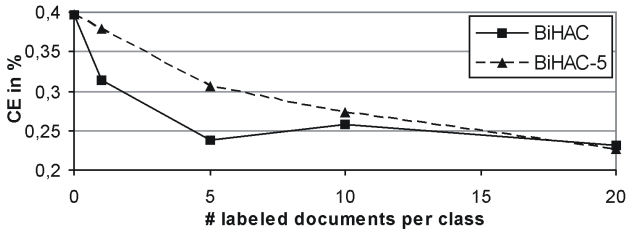

Figure 5. Cluster Error

Figure 6 presents the F-score. It is a class specific measure. In order to get an overall evaluation of the performance, we calculate the mean over all classes. Since we have a hierarchical setting here, we calculate two different mean values, one for each hierarchy level. On the first level, we only distinguish between the three classes Banking, Programming, and Sport. Documents labeled with a subclass are counted to be labeled with its parent class. The second hierarchy level consists of the eight subclasses.

Both algorithms show a similar behavior. The F-score is increasing with increasing amount of labeled data on both hierarchy levels. The performance of both algorithms is higher on level 1. That means, both algorithms make most mistakes between classes that describe subtopics but can separate the main classes quite well. This is also expected as distinguishing between top-level classes is easier then between more specific classes. The results obtained after only

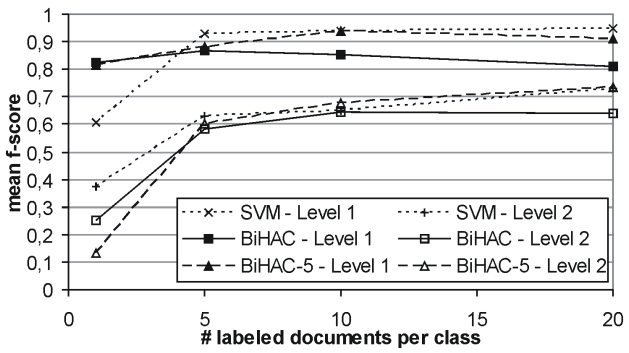

Figure 6. Mean F-Score

five iterations of BiHAC are comparable to SVM or even better. However, optimal adaptation to the user specific constraints leads to a decrease of performance. This is probably due to overfitting to the training data. We will tackle this problem in future work. It is interesting to mention that our algorithm clearly outperforms SVM on the first hierarchy level, when only a single labeled document is available.

Our evaluation has shown two main points. First, our algorithm is actually able to adapt towards a user specific structure. And second, classification performance gained by a very simple labeling strategy provides results comparable or even better to state of the art classifiers as SVM.

Furthermore, we like to mention two more aspects, which were not further evaluated. On the one hand, the algorithm provides further structuring capabilities. This includes refinement of user classes or the exploration of new hidden classes. On the other hand, the personalization of the clustering is gained by learning term weights. The final clustering – as presented to the user – is influenced only by using these weights in the similarity measure. The clustering process itself is not changed and thus the obtained weighting scheme could also be stored in a user profile for further use, e.g. to describe a user's interests or to be applied to other different classification or clustering methods. To evaluate these aspects is subject to future work.

5. Conclusion and Future Work

In this paper, we presented an approach for user-specific hierarchical clustering called BiHAC. As a basis, we extended the definitions of constraint based clustering to express hierarchical constraints. BiHAC proved to be able to adapt to a user structure by learning term weights in an iterative process. This adaptation as well as classification performance were evaluated using a benchmark dataset. The results encourage further work in this direction. Besides the issues already mentioned throughout the paper, we want to add a post-processing step to our clustering algorithm that extracts a more coarse-grained cluster structure from the fine-grained dendrogram containing meaningful clusters. This is especially important for result visualization as a dendrogram requires too much navigation until a sought document is found. Furthermore, we want to explore the integration of more than one given hierarchy, e.g. from different users, to further guide the clustering process.

References

[1] S. Basu, A. Banerjee, and R. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proc. of the 4th SIAM International Conf. on Data Mining*, 2004.

[2] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of the 7th Intern. Conf. on Database Theory*, pages 217–235, 1999.

[3] C. Borgelt and A. Nürnberger. Fast fuzzy clustering of web page collections. In *Proc. of PKDD Workshop on Statistical Approaches for Web Mining (SAWM)*, 2004.

[4] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[5] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. Technical Report TR2003-1892, Cornell University, 2003.

[6] A. Hotho, A. Nürnberger, and G. Paaß. A brief survey of text mining. *GLDV-Journal for Computational Linguistics and Language Technology*, 20(1):19–62, 2005.

[7] H. Kim and S. Lee. An effective document clustering method using user-adaptable distance metrics. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 16–20, New York, NY, USA, 2002. ACM Press.

[8] D. Klein, S. Kamvar, and C. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proc. of the 19th International Conf. on Machine Learning*, pages 307–314, 2002.

[9] A. Nürnberger and M. Detyniecki. Weighted self-organizing maps - incorporating user feedback. In *Proc. of the joined 13th International Conference on Artificial Neural Networks and Neural Information Processing*, pages 883–890, 2003.

[10] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[11] G. Salton, A. Wong, and C. Yang. A vector space model for information retrieval. *Journal of the American Society for Information Science*, 18(11):613–620, 1975.

[12] M. Sinka and D. Corne. A large benchmark dataset for web document clustering. In *Soft Computing Systems: Design, Management and Applications, Vol. 87 of Frontiers in Artificial Intelligence and Applications*, pages 881–890, 2002.

[13] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proc. of 18th Intern. Conf. on Machine Learning*, 2001.

[14] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512. 2003.