

User Modelling for Interactive User-Adaptive Collection Structuring

Andreas Nürnberger and Sebastian Stober

Institute for Knowledge and Language Engineering,
Faculty of Computer Science
Otto-von-Guericke-University Magdeburg, D-39106 Magdeburg, Germany
{nuernb,stober}@iws.cs.uni-magdeburg.de

Abstract. Automatic structuring is one means to ease access to document collections, be it for organization or for exploration. Of even greater help would be a presentation that adapts to the user's way of structuring and thus is intuitively understandable. We extend an existing user-adaptive prototype system that is based on a growing self-organizing map and that learns a feature weighting scheme from a user's interaction with the system resulting in a personalized similarity measure. The proposed approach for adapting the feature weights targets certain problems of previously used heuristics. The revised adaptation method is based on quadratic optimization and thus we are able to pose certain constraints on the derived weighting scheme. Moreover, thus it is guaranteed that an optimal weighting scheme is found if one exists. The proposed approach is evaluated by simulating user interaction with the system on two text datasets: one artificial data set that is used to analyze the performance for different user types and a real world data set – a subset of the banksearch dataset – containing additional class information.

1 Introduction

In many domains users are interested in information that they cannot clearly specify, e.g. by some keywords. For example, a journalist might be researching the background of his current article or someone might look for new music that fits his taste. In such everyday exploratory search scenarios, usually large data collections are accessed. Structuring is one means to ease these tasks, especially if it reflects the user's personal interests and thus is intuitively understandable. Unfortunately, except for the user's own collections, personalized structuring is not generally available.

In previous work, we presented an user-adaptive retrieval system considering the task of organizing an unstructured collection. An initially unpersonalized structure is learned with a growing self-organizing map which provides an overview of the collection. The user can then work with this representation, search and access objects and move items on the map that he feels should be located elsewhere. From this user interaction, an individual feature weighting scheme for similarity computation that represents the user's organization preferences is continuously derived and used to adapt the collection organization.

Previously, we used greedy heuristics to learn this feature weighting scheme [7, 5, 6]. Using these heuristics, however, caused several problems: Since the values of the feature weights could not be limited, extreme weighting schemes often occurred. Furthermore, the heuristics could not formally guarantee that all manually moved objects are assigned to their target, and no additional constraints could be formulated that could e.g. increase the interpretability of the result.

To address these problems, we have revised our adaptation method by using quadratic optimization to derive the feature weights. This allows the introduction of additional constraints on the feature weights and moreover guarantees to find an optimal weighting scheme if it exists.

The remainder of this paper is organized as follows: Section 2 reviews crucial aspects of the user-adaptive retrieval prototype. In Section 3, the new method for weight adaptation is introduced. Subsequently, the experimental setup for evaluation is explained in Section 4. Section 5 presents and discusses the results of the experiments. Finally, we draw conclusions and point out directions for future research in Section 6.

2 An Interactive Retrieval System based on a Self-Organizing Map

In the following we review our adaptive retrieval system that we previously introduced in [7]. We first describe, how documents are preprocessed and represented. Section 2.3 outlines, how the initial clustering is computed. Finally, the generic principle of the adaptation algorithm is explained in Section 2.4.

2.1 Document Preprocessing and Representation

To be able to cluster text document collections, we have to map the text files to numerical feature vectors. Therefore, we first applied standard preprocessing methods, i.e., stopword filtering and stemming (using the Porter Stemmer [8]), encoded each document using the vector space model [11] and finally selected a subset of terms as features for the clustering process with a greedy heuristics. These processing are briefly described in the following.

In the vector space model text documents are represented as vectors in an m -dimensional space, i.e., each document j is described by a numerical feature vector $\mathbf{x}_j = (x_{j1}, \dots, x_{jm})$. Each element of the vector represents a word of the document collection, i.e., the size of the vector is defined by the number of words of the complete document collection.

For a given document j x_{jk} defines the importance of the word k in this document with respect to the given document collection C . Large feature values are assigned to terms that are frequent in relevant documents but rare in the whole document collection [10]. Thus a feature x_{jk} for a term k in document j is computed as the term frequency tf_{jk} times the inverse document frequency idf_k , which describes the term specificity within the document collection.

In [9] a scheme for computation of the features was proposed that has meanwhile proven its usability in practice. Besides term frequency and inverse document frequency (defined as $\text{idf}_k = \log(n/n_k)$), a length normalization factor is used to ensure that all documents have equal chances of being retrieved independent of their lengths:

$$x_{jk} = \frac{\text{tf}_{jk} \log \frac{n}{n_k}}{\sqrt{\sum_{l=1}^m (\text{tf}_{jl} \log \frac{n}{n_l})^2}}, \quad (1)$$

where n is the size of the document collection C , n_k the number of documents in C that contain term k , and m the number of terms that are considered.

Based on this scheme a document j is described by an m -dimensional vector $\mathbf{x}_j = (x_{j1}, \dots, x_{jm})$ of term features and the similarity S of two documents (or the similarity of a document and a query vector) can be computed based on the inner product of the vectors (by which—if we assume normalized vectors—the cosine between the two document vectors is computed), i.e.

$$S(\mathbf{x}_j, \mathbf{x}_k) = \sum_{l=1}^m x_{jl} \cdot x_{kl}. \quad (2)$$

For a more detailed discussion of the vector space model and weighting schemes see, for instance, [1, 10, 11].

2.2 Index Term Selection

To reduce the number of words in the vector description we applied a simple method for keyword selection by extracting keywords based on their entropy. For each word k in the vocabulary the entropy as defined by [4] was computed:

$$W_k = 1 + \frac{1}{\log_2 n} \sum_{j=1}^n p_{jk} \log_2 p_{jk} \quad \text{with} \quad p_{jk} = \frac{\text{tf}_{jk}}{\sum_{l=1}^n \text{tf}_{lk}}, \quad (3)$$

where tf_{jk} is the frequency of word k in document j , and n is the number of documents in the collection. Here the entropy gives a measure how well a word is suited to separate documents by keyword search. For instance, words that occur in many documents will have low entropy. The entropy can be seen as a measure of the importance of a word in the given domain context. As index words a number of words that have a high entropy relative to their overall frequency have been chosen, i.e. of words occurring equally often those with the higher entropy can be preferred. Empirically this procedure has been found to yield a set of relevant words that are suited to serve as index terms [3].

However, in order to obtain a fixed number of index terms that appropriately cover the documents, we applied a greedy strategy: From the first document in the collection the term with the highest relative entropy is selected as an index term. Then, this document and all other documents containing this term are marked. From the first of the remaining unmarked documents again the term

with the highest relative entropy is selected as an index term. Then again, this document and all other documents containing this term are marked. This is repeated until all documents have been marked. Subsequently, the documents are unmarked and the process is started all over again. It can be terminated when the desired number of index terms have been selected.

2.3 Initial Clustering

Representing documents as described in the preceding section and using the cosine similarity measure, documents can be clustered by the growing self-organizing map approach presented in [7]. The algorithm starts with a small initial grid composed of hexagon cells, where each hexagon refers to a cluster and is represented by a randomly initialized prototype feature vector. Each document is then assigned to the most similar cluster in the grid resulting in an update of the respective prototype and to some extent of the surrounding prototypes (depending on a neighborhood function). Having all documents assigned to the grid, the inner cluster distance can be computed. The clusters where it exceeds some predefined threshold are split resulting in a grown map. This process is repeated until no more cells need to or can be split or an empty cell occurs (i.e. a cluster with no assigned documents). It results in a two-dimensional topology preserving the neighborhood relations of the high dimensional feature space. I.e. not only documents within the same cluster are similar to each other but also documents of clusters in the neighborhood are expected to be more similar than those in more distant clusters. Using such a growing approach ensures that only as many clusters are created as are actually needed.

2.4 User-Adaptivity

The algorithm outlined in Section 2.3 works in an unsupervised manner. It depends only on the choice of the features for representation and the initial similarity measure how documents are grouped. However, the user has often a better intuition of a “correct” clustering. In cases where he does not agree on the cluster assignment for a specific document, the user can change it by dragging the document from a cell and drop it onto a different cell of the grid. In online mode, the system will immediately react and modify the map, whereas in batch mode it will wait until the user manually triggers the modification after he has made several reassignments.

The remapping is performed by introducing feature weights that are used during the similarity computation. The basic idea is to change the feature weights for the similarity computation in such a way that the documents moved by the user are now assigned to the desired clusters, i.e. the prototypes defining the new cluster centers are more similar to the moved objects than the prototypes defining the original cluster centers.

While in prior work we used greedy heuristics to learn local or global weights for each feature, i.e. weights that influence the importance of a term with respect to the whole map or just with respect to a cell (see, e.g., [7, 5, 6]), we propose in

this work an optimization approach based on quadratic optimization as described in the following. This approach works much more stable than the heuristics and it is also possible to define constraints on the weight vector.

3 Quadratic Optimization

The problem of dragging documents as described above can be mapped to a problem of cluster reassignment: Assume a document d that is assigned to a cluster c_1 , i.e. for the similarity holds

$$\text{sim}(c_1, d) < \text{sim}(c_i, d) \quad \forall i \neq 1. \quad (4)$$

If c_1 is dragged by a user to a cluster c_2 we have to adapt the underlying similarity measure such that now

$$\text{sim}(c_2, d) < \text{sim}(c_i, d) \quad \forall i \neq 2 \quad (5)$$

holds. If we assume that a user reassigns more than one document, a similar constraint has to be defined for each reassigned document. In the following, this process is described more formally.

3.1 Formalization

In order to solve the reassignment problem as described above using a quadratic problem solver, we have to formulate our problem at hand appropriately. Therefore, we make the following assumptions:

Let $\mathbf{w} := (w_0, w_1, \dots, w_m)$ be the weight vector used during similarity computation, \mathbf{x}_j be a document vector describing a document d_j as defined above and \mathbf{s}_j be a vector defining a cluster prototype. Then we can define a weighted similarity computation between document and prototype as follows:

$$S(\mathbf{x}_j, \mathbf{s}_k) = \sum_{l=1}^m x_{jl} \cdot w_l \cdot s_{kl}. \quad (6)$$

Thus, we can modify the influence of specific features (here words) during similarity computation. If we initialize all weights w_l with 1 we obtain the standard inner product as defined above.

The task of our optimization problem is to increase the similarity of a document to a target prototype \mathbf{t}_i (this is the cluster cell to which a user has moved a document) such that this similarity is bigger than the similarity to all other prototypes \mathbf{s}_k , in particular to the prototype of the cell to which the document was initially assigned. Using the weighted similarity measure, this means that we have to change the weights such that

$$\sum_{l=1}^m x_{jl} \cdot w_l \cdot t_{il} > \sum_{l=1}^m x_{jl} \cdot w_l \cdot s_{kl} \quad \forall k \neq i. \quad (7)$$

Note that the change of the weight vector \mathbf{w} should be as small as possible in order to avoid too much corruption of the initial cluster assignments. Therefore, we demand that the sum over all (quadratic) deviations of the weights from their initial value 1 should be minimal (Eq. 8) and further that the weights should be non-negative (Eq. 9) and the sum of all weights should always be m (Eq. 10). If we furthermore assume, that we like to ensure that several documents can be reassigned at the same time, we can now formulate this problem as a quadratic optimization problem, i.e. we are looking for weights, such that we minimize

$$\min_{\mathbf{w} \in \mathbb{R}^m} \sum_{l=1}^m (w_l - 1)^2 \quad (8)$$

subject to the constraints

$$w_l \geq 0 \quad \forall 1 \leq l \leq m \quad (9)$$

$$\sum_{l=1}^m w_l = m \quad (10)$$

$$\begin{aligned} \sum_{l=1}^m x_{jl} \cdot w_l \cdot t_{1l} &> \sum_{l=1}^m x_{jl} \cdot w_l \cdot s_{kl} & \forall k \neq i \\ \sum_{l=1}^m x_{jl} \cdot w_l \cdot t_{2l} &> \sum_{l=1}^m x_{jl} \cdot w_l \cdot s_{kl} & \forall k \neq i \\ &\vdots \\ \sum_{l=1}^m x_{jl} \cdot w_l \cdot t_{rl} &> \sum_{l=1}^m x_{jl} \cdot w_l \cdot s_{kl} & \forall k \neq i. \end{aligned} \quad (11)$$

Here, Eq. 10 prevents very large weight values and Eq. 11 ensures that all r documents that were moved by a user are assigned to the cluster cell specified by the user.

3.2 Remarks

The changes of the underlying similarity measure based on the user's mappings might lead to a reassignment of further documents. This is intended, since we assume that the resulting weight changes define the structuring criteria of the user. This way, moving just a small number of object manually, many others could be relocated automatically as a result of the adaption which saves additional effort for the user.

The constraints defined by Eq. 11 can also be used to fix documents to specific clusters, i.e. prevent them from being moved to other cluster cells during user interaction with the system and the resulting weight adaptations. This can be especially useful, if a user likes to ensure that typical objects are used as stable reference landmarks in the mapping.

4 Experimental Setup

We conducted two experiments in which we simulate the user interaction with the system. The general approach chosen is as follows:

```
modify objects by adding random features
learn map on modified objects
repeat
  select an object o to be moved
  select most similar cell c for o according to user
  move o to c
until o could not be moved
```

Given a set of objects described by some specific features, we assume that a user would consider all of these features to be equally important and ignore any other features when comparing the objects. A similarity measure according to the user’s preferences would then just compare the relevant features. Consequently, the initial similarity measure on the object set (weighting all features equally) is regarded as ground truth.

In the first step, we add some “noise” to the object descriptions by adding random features.¹ Because the GSOM learning algorithm would ignore noisy features that do not contain any information, the random feature values are obtained as follows: Assuming that n features are to be added, random prototype vectors for n groups are generated. These vectors are randomly initialized with zeros and ones, where the probability for a one is $\log n/n$. Afterwards, a gaussian random number with a mean of 0.2 and standard deviation of 1.0 is added, thus “blurring” the prototype vectors. For each object to be modified, a prototype vector is randomly selected. The feature values to be added to the object’s description are then generated by (again) adding a gaussian random number with a mean of 0.2 and standard deviation of 1.0 to the respective feature value of the prototype vector. This empirically derived procedure results in enough information to significantly influence the map learning algorithm.

As a result of the added noise, the assignment of the objects to the map differs from the simulated user’s point of view. Iteratively, he selects an objects on the map, moves it to the best position according to his ground truth similarity measure and lets the map update its feature weights. This process is repeated until the selected object could not be moved because it is already at the desired position or due to limitations of the adaptation algorithm that we will discuss later in Section 5.1.² In the following, we give details on the specific experiments, particularly on the datasets, the methods for object selection and the evaluation measures used.

¹ Just masking out features instead of adding random ones does not work. Due to the sparse nature of the document vectors, masking only few index terms already results in documents that cannot be compared according to the user’s preferences because they have no index terms anymore. However, to provide the adaption algorithm with enough “room” for improvement, it would be necessary to mask many index terms.

² It can happen very early in the process that the selected object is already at its best position. Therefore, we checked not only one but up to 1% of all objects on the map.

4.1 Experiment 1

The first experiment was conducted on the dataset that has been already used in [3] It comprises 1914 text documents taken from a news archive that covers several scientific topics. The documents in the dataset had been already pre-processed and 800 index terms had been selected using the method described in Section 2.2.

For selection of the object to move, we distinguish between the following four scenarios of user action simulation whose relations are depicted in Table 1:

1. *Greedy selection of cell and object:* First, the cell with the lowest average pairwise (ground truth) similarity of the contained objects is chosen for further investigation. Within this cell, the object with the lowest average pairwise (ground truth) similarity with all other objects in the same cell is selected to be moved.
2. *Greedy selection of cell and random selection of object:* The cell is chosen as in the previous scenario. However, an arbitrary object is selected from this cell.
3. *Random selection of cell and greedy selection of object:* Here, the cell is chosen randomly whereas the object to be moved is selected from this cell by the greedy approach used in scenario 1.
4. *Random selection of cell and random selection of object:* In this scenario, both, the cell and the object to be moved from the cell, are selected randomly.

Note that scenario 3 appears to be the one that comes closest to the real use case where a user does not look into all cells before picking an object to be moved but within a specific cell tends to select the object that fits least into the cell according to his preferences.

For the evaluation of the adaptation algorithm, we utilize the average *top-10 precision* of the similarity measure on the object set: A ranked top-10 list of similar objects is computed for each object according to the system’s current similarity measure and compared with a ground truth top-10 list. The percentage of matches is computed – ignoring the order of the ranking – and averaged over all objects. This measure resembles a user’s natural way of assessing a retrieval result where the correct order of relevance is less important than the actual number of highly relevant objects within the first few items of a result list.

		cell selection	
		greedy	random
object selection	greedy	scenario 1	scenario 3
	random	scenario 2	scenario 4

Table 1. Relations of the four scenarios for user simulation.

4.2 Experiment 2

In this experiment we use a subset³ of the Banksearch dataset [12], a pre-classified dataset containing 11,000 web pages from 11 different categories grouped into 4 high-level topics (finance, programming, science, sports). The dataset used in this experiment has been constructed as follows: 100 documents (i.e. 10%) were selected for each of the 11 categories. The documents were preprocessed with the same methods as the dataset in experiment 1. After index term selection, documents with less than 5 index terms were removed resulting in a set of 947 documents described by 800 index terms.

In contrast to the first dataset, each document additionally contains information about its topic class. This information is not used during training of the map, however it is utilized to select the object to be moved. Analogously to the first experiment, we regard four scenarios but replace the greedy heuristic by one that is based on the class information: For each cell the majority and minority class(es) are determined and the cell with the highest difference in the frequencies of these classes is selected. Further, during the object selection step only those objects belonging to a minority class of the cell are considered. Finally, only those cells can be selected as target cell that have a majority class that matches the class of the object to move.

This experiment resembles a use case where the user tries to sort objects according to (subjective) classes unknown to the system. Apart from the top-n precision that has also been applied here, we use the well-known evaluation measures *purity*, *inverse purity* and *f-measure* as e.g. described in [2]. These measures return values in the interval $[0, 1]$, with 1 being the optimal value. The purity measures the homogeneity of the resulting clusters and is 1, iff for each cluster all contained objects belong to the same class. Whereas, the inverse purity – also known as *microaveraged precision* – measures how stable the pre-defined classes are when split up into clusters. It is 1, iff for each class all belonging objects are grouped into a single cluster. The F-measure is similar to the inverse purity. However, it punishes overly large clusters, as it includes the individual precision of these clusters into the evaluation.

5 Results

5.1 Experiment 1

The results for experiment 1 are shown in Figure 1. For all scenarios and noise levels the top-10 precision increased significantly to a value between 0.82 and 0.97 with a mean of 0.93. As expected, with an increasing level of noise in the data, more objects need to be moved manually in order to reach an acceptable degree of adaptation, yet – except for the highest noise level (100 random features) – moving at most 1% of all objects was sufficient. This may, however, be very much dependent on the object representation used. For text retrieval, feature vectors are very sparse. This is probably disadvantageous for the adaptation algorithm,

³ A subset was taken to reduce the computational effort of the evaluation.

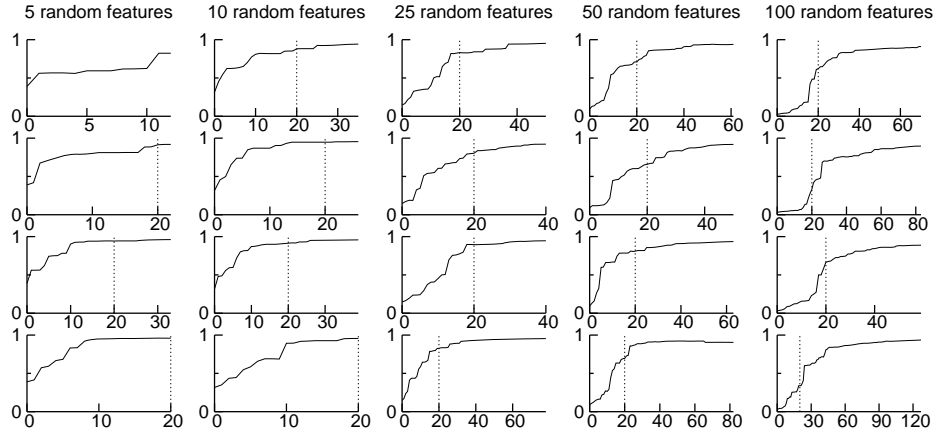


Fig. 1. Top-10 precision for experiment 1 with varying noise levels (number of random features) against the number of iterations (number of manually moved objects) for the four user simulation scenarios (from top to bottom): (1) Greedy selection of cell and object, (2) Greedy selection of cell and random selection of object, (3) Random selection of cell and greedy selection of object, and (4) Random selection of cell and object. The dotted vertical line at 20 iterations marks the point where 1% of the collection has been moved manually by the simulated user.

as to adapt a feature’s weight, the feature needs to be present (i.e. non-zero) in a moved object. Using feature vectors that are less sparse could result in faster adaptation.

Surprisingly, the scenarios that used partly or fully random selection of the object to move did not yield significantly worse results than the fully greedy one. In some cases their maximum top-10 precision was even slightly better and sometimes they converged faster to a good adaptation. Mostly, however, they did not terminate as quickly, but this is not a problem because it only refers to the simulation method and not to the adaptation algorithm itself.

On the other hand, the simulation sometimes terminated a little too early, leaving still room for improvement as e.g. can be seen for scenario 1 with 5 or 10 random features. In about half of the cases the simulated user just did not find any object to move (within the 1% analyzed). However, for the other half, the adaptation algorithm was not able to compute new weights because adding the constraints for the object to move caused an inconsistency in the system. I.e. there is no feature weighting scheme such that the manually moved objects are assigned to the map as desired by the user. This is a limitation of the proposed adaptation algorithm that needs to be overcome in some way to not put the user acceptance of the system at risk.

Further, a side-effect could be observed, that we want to illustrate by an example, where the user simulation terminated early because of an inconsistency in the system: Figure 2 shows the changes of the feature weights for scenario 1 with 10 random features. The mean weight of the random features decreases to values close to zero but still leaving some non-zero weights because of the early termination. At the same time, the mean weight of the other features slightly

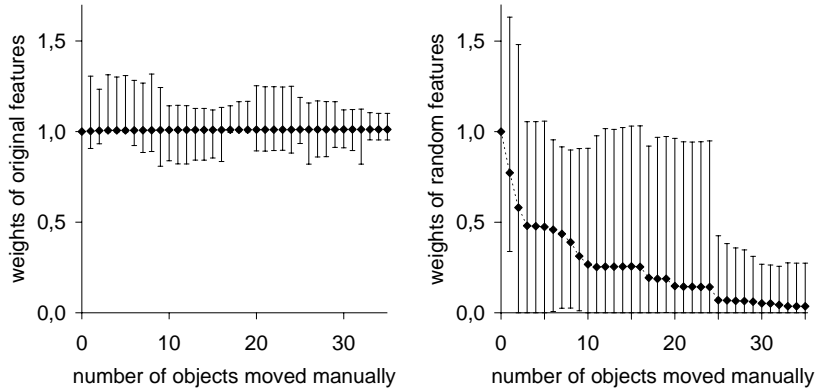


Fig. 2. Weights (mean, maximum and minimum) for original (left) and added random (right) features after each iteration for scenario 1 (greedy selection of cell and object) with 10 random features.

increases due to the additional weight mass from the random features. However, as the minima and maxima indicate, there are always some extreme weights. Figure 3 shows a histogram of the weights for the original (i.e. non-random) features after the last iteration. The weights that differ significantly from the mean refer to index terms of the manually moved objects. They were especially “emphasized” by the adaptation algorithm as only they can be used to decide on the assignment of the manually moved objects to the map. Consequently, the resulting histogram in Figure 3 differs from the ideal one where all weights are equal. However, as the adaptation algorithm favors weights close to 1.0 that difference is rather small. It is further possible, that the values of the random features to some extent correlate with the non-random ones and accidentally capture some aspects of subgroups. Though this is rather unlikely in general, it may cause side-effects in specific cases. However, this is caused by the evaluation method and thus will not occur in real world applications.

5.2 Experiment 2

In this experiment, we again obtained similar results for all scenarios. Thus, we confine ourselves in the following on the discussion of the results for scenario 1 (i.e. greedy object selection). Figure 4 shows the results on the second dataset

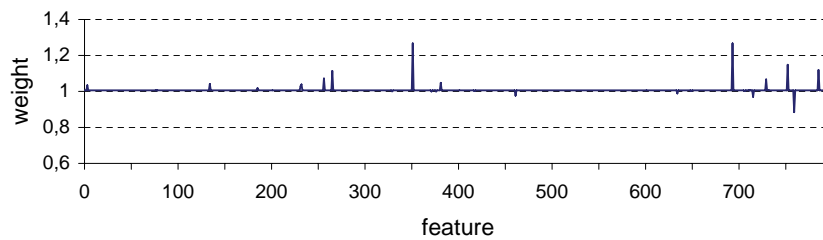


Fig. 3. Histogram of the weights for the original features after the last iteration for scenario 1 (greedy selection of cell and object) with 10 random features.

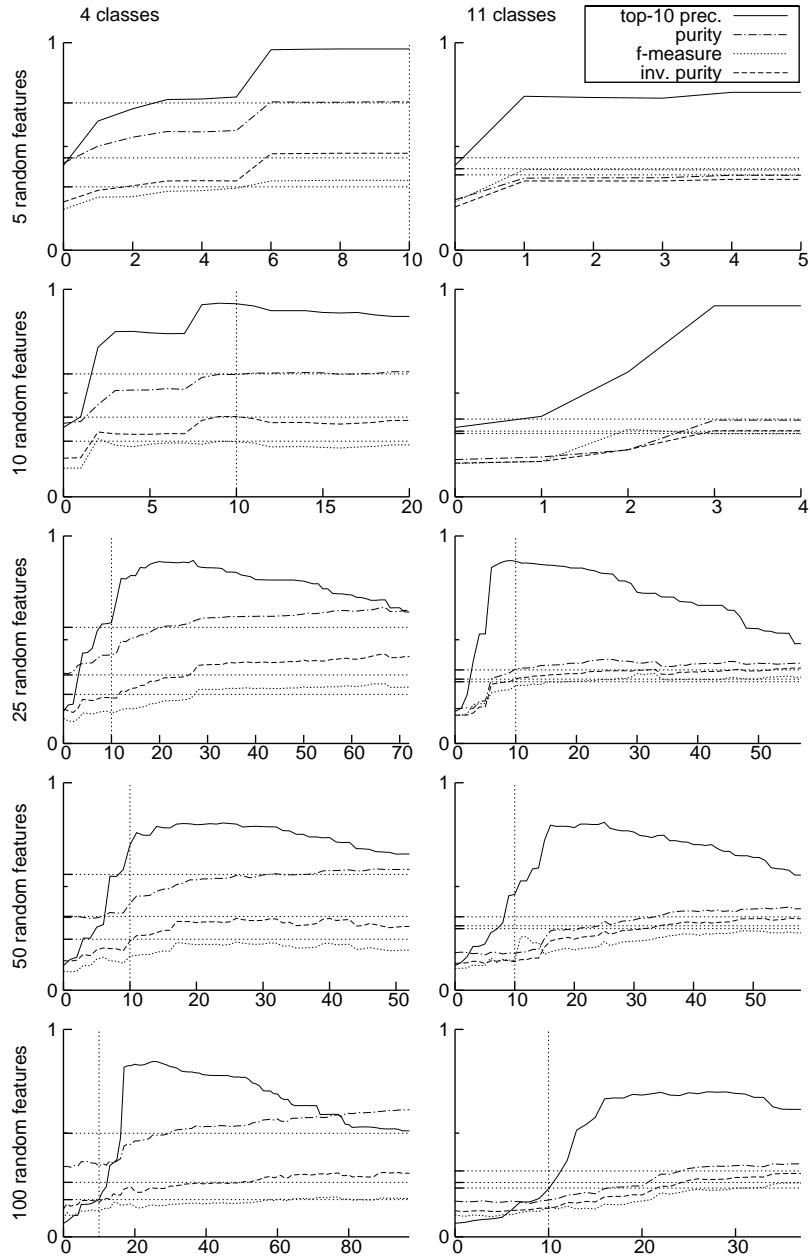


Fig. 4. Performance of the adaption method in experiment 2, using the 4 top-level topics and the 11 low-level categories of the banksearch dataset as classes, with varying noise levels (number of random features) plotted against the number of iterations (number of manually moved objects). Cell and object were selected greedily (scenario 1). The three dotted horizontal marker lines depict (from top to bottom) the baseline values for purity, f-measure and inverse purity. The dotted vertical line at 10 iterations marks the point where 1% of the collection has been moved manually by the simulated user.

using the 4 top-level topics and the 11 low-level categories of the banksearch dataset as classes. In all test cases the final value for purity, inverse purity and f-measure came close to or exceeded the baseline value, that was obtained by setting the weights of the random features to 0 and equally distributing the weight mass on the remaining feature weights. The baseline values could be exceeded by some extent through adaptation because here the additional class information was used which was unknown to the map learning algorithm that solely worked on the index terms as features. Especially, the greedy heuristic applied for object selection favours cells with low purity. On the other hand, the top-10 precision only increases up to some point. From there it slowly decreases again. This is because the weight adaptation is not approximating a map that was learned only on the original objects (i.e. without the noise) what would optimize top-10 precision, but instead, tries to separate the classes.

6 Conclusions

In this contribution we presented a method for feature weight adaptation in order to learn user specific structuring criteria. The approach is based on quadratic optimization and ensures that an optimal weighting scheme can be found if it exists. Furthermore, we evaluated the proposed method on two datasets by simulating the interaction of a user with the system. Therefore, we modelled different user types that simulated typical and worst case behaviors. We have shown that the system was able to derive appropriate solutions for all scenarios.

In future work we will modify the optimization strategy such that at least a sub-optimal solution is returned if no optimal one can be found. This is especially important for scenarios where the users structuring criteria can not be represented by the available document features.

References

1. W. R. Greiff. A theory of term weighting based on exploratory data analysis. In *21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, 1998. ACM.
2. 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.
3. A. Klose, A. Nürnberger, R. Kruse, G. K. Hartmann, and M. Richards. Interactive text retrieval based on document similarities. *Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy*, 25(8):649–654, nov 2000.
4. K. E. Lochbaum and L. A. Streeter. Combining and comparing the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing and Management*, 25(6):665–676, 1989.
5. A. Nürnberger and M. Detyniecki. Weighted self-organizing maps - incorporating user feedback. In *Artificial Neural Networks and Neural Information Processing - ICANN/ICONIP 2003, Proc. of the joined 13th Int. Conf.*, 2003.
6. A. Nürnberger and M. Detyniecki. Externally growing self-organizing maps and its application to e-mail database visualization and exploration. *Applied Soft Computing*, 6(4):357–371, 2006.

7. A. Nürnberger and A. Klose. Improving clustering and visualization of multimedia data using interactive user feedback. In *Proc. of the 9th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'02)*, 2002.
8. M. Porter. An algorithm for suffix stripping. *Program*, pages 130–137, 1980.
9. G. Salton, J. Allan, and C. Buckley. Automatic structuring and retrieval of large text files. *Communications of the ACM*, 37(2):97–108, Feb 1994.
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. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. (see also TR74-218, Cornell University, NY, USA).
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.