STAIRS 2016
D. Pearce and H.S. Pinto (Eds.)

doi:10.3233/978-1-61499-682-8-51

OHDOCLUS – Online and Hierarchical Document Clustering

Rui ENCARNAÇÃO[1] and Hugo Gonçalo OLIVEIRA
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal

Abstract. Usually, clustering algorithms consider that document collections are static and are processed as a whole. However, in contexts where data is constantly being produced (e.g. the Web), systems that receive and process documents incrementally are becoming more and more important. We propose OHDOCLUS, an online and hierarchical algorithm for document clustering. OHDOCLUS creates a tree of clusters where documents are classified as soon as they are received. It is based on COBWEB and CLASSIT, two well-known data clustering algorithms that create hierarchies of probabilistic concepts and were seldom applied to text data. An experimental evaluation was conducted with categorized corpora, and the preliminary results confirm the validity of the proposed method.

Keywords. Clustering, document clustering, online clustering, hierarchical clustering, dimensionality reduction

1. Introduction

Clustering is the division of objects into clusters or "the art of finding groups in data" [1]. A clustering algorithm has to decide which clusters must be created and to which cluster(s) each object must be assigned.

Document clustering aims at the automatic organization of a set of documents into clusters. It is used in many tasks: to organize search results, to browse collections of documents, to improve information retrieval systems, to detect new topics in news streams, and even to ascribe authorship or detect plagiarism. The importance of automatic tools to organize collections of documents has increased in recent years since the number of electronic documents has grown much faster than our ability to use them.

These clustering algorithms can be classified in many ways [2,3,4] but we will focus on two. Depending on the number of levels of the clustering, algorithms can be classified in *hierarchical,* if they create a tree of clusters, or *partitioning* if they create a flat partition of clusters. According to the way the documents are processed, algorithms can be classified as *batch* (if the documents are processed as a whole) or *online* (if documents are processed sequentially, one at a time).

We aim to develop a document clustering system that is as automatic as possible, without the need to define a priori the number of clusters or other setup parameters, and able to provide, in each moment, an updated hierarchy of the received documents.

[1] This work was financed by national funding via the Foundation for Science and Technology, by the European Regional Development Fund (FEDER), through the COMPETE 2020 - Operational Program for Competitiveness and Internationalization (POCI), and by the FCT scholarship grant SFRH/BD/73543/2010.

In that vein, our proposed system, OHDOCLUS, is hierarchical and online. Hierarchical to enable browsing the document collection and to produce results with different detail for users with different needs; this option also bypasses the need of prespecify the number of clusters, allowing the algorithm to choose the best number in each case, and allowing the user to cut off the final tree at any level. We choose an online algorithm because document repositories (tweets, blog posts, product reviews, document folders, book collections) are becoming increasingly online, with constant updates, and even the offline collections can be updated anytime. In the current Web, new documents are constantly being published (e.g. social networks or news feeds).

The main contributions of this work are: an online and hierarchical clustering algorithm; the adaptation and combination of two existing data clustering algorithms to document clustering; and the definition of a new framework for online document clustering, with the development of better tree reorganization mechanisms to enable a continuous improvement of the clustering quality.

In the next section, we present some related work. In section 3, we describe our algorithm in more detail, and the choices made related to document representation, dimensionality reduction, and clustering evaluation. Then, we present some preliminary experimentation done with a prototype of OHDOCLUS, and we end with some possible paths for future research.

2. Related Work

Here, we review the main clustering algorithms related to the proposed system. We start with a brief reference to the two most important clustering types (partitional and hierarchical) and move on to other relevant concepts, namely conceptual clustering, online document clustering, and topic modeling.

2.1. Partitional and Hierarchical Clustering

K-means [5] is the most widely used clustering algorithm. It builds a flat partition with a predefined number (K) of clusters, based on the distance between the instances. Clusters are represented by their centroids. K-means starts with K random initial centroids and assigns each item to its closest centroid. Then, new centroids are iteratively calculated from the objects assigned to each cluster, and assignments are changed if necessary. The algorithm stops when convergence is obtained, and no change occurs.

The main advantages of K-means are its simplicity and efficiency when compared to other techniques, particularly if the dataset is large. One disadvantage is the need to specify the number of desired clusters, which may lead to suboptimal results. Moreover, the algorithm is too sensitive to the initial seeds and outliers.

Hierarchical clustering [6] creates a tree of clusters, with the top node containing all the items, and the leaves containing only one item. Depending on the direction of construction, these methods can be classified as *divisive* - start with one node with all the documents and split it until getting single nodes - or *agglomerative* - start with single nodes and join the most similar pair until the most general cluster is created.

These algorithms have no mechanism to recover from bad decisions while choosing the nodes to merge or split, and this can have an impact in terms of clustering accuracy. Furthermore, due to the complexity of computing the similarity between every pair of clusters, they are typically not scalable for large datasets [6,7].

2.2. Conceptual Clustering

Conceptual clustering algorithms are methods that generate a concept description for each cluster. Most of them are incremental and build a hierarchy of clusters that contain a probabilistic concept that describes the objects assigned to the cluster [8]. COBWEB [9] and CLASSIT [10] are two well-known algorithms for conceptual data clustering.

COBWEB accepts items described by nominal attributes and builds a hierarchy where each node stores the conditional probability of every value for each attribute. Unlike most algorithms that are based on the distance between objects, COBWEB uses category utility [11] as the evaluation function. *Category utility* is a heuristic trade-off between intra-class similarity and inter-class dissimilarity. The category utility of a clustering is the expected increase in the number of attribute values that can be correctly guessed using the clustering over the expected correct guesses without it.

When classifying an object, the algorithm traverses the tree top-down, considering, in each node, four possible operations: (a) insert the object in the best child cluster; (b) create a new cluster; (c) merge two children; (d) split a child. The algorithm computes the category utility of the hierarchy generated by each operator and applies the best one.

CLASSIT is a successor of COBWEB and overcomes some of its limitations. It accepts objects described only by numeric features. It uses the same operators but introduces two parameters: *cutoff*, which stops the classification in a node whenever the gain in category utility is below that threshold; and *acuity*, a parameter that imposes a minimum for the standard deviation (σ) of a feature, to bypass the problem of null deviations in a cluster, since the category utility for numeric features is proportional to $1/\sigma$.

We can consider that COBWEB and CLASSIT execute a hill-climbing search in the space of concept hierarchies guided by the category utility. The two backtracking operators (merge and split) enable the algorithm to recover from wrong decisions, making it less sensitive to the order of instances. These systems satisfy the criteria proposed for evaluating incremental clustering [9]: the cost of incorporating each object is low, the hierarchies produced have high quality and promote correct predictions.

2.3. Online Document Clustering

It is important to clarify that, in the scope of this work, *online document clustering* implies that the document collection is not available from the beginning. Instead, documents arrive and are processed sequentially, one by one. The number of documents is unknown and (potentially) unbounded. In these systems, there is usually time and memory restrictions that require the document processing to be fast and also with heavy restrictions on storage.

Incremental is frequently used as a synonym of online (we used it in an initial stage of this research) but sometimes it means that documents are processed in small batches. *Document stream clustering* is sometimes used with a similar meaning, but usually, there is the assumption that older documents are of less importance, and to that end, they use a decay rate or a sliding window to “forget” oldest documents [12].

The first experiments with online document clustering appeared in the field of topic modeling, to detect events in news streams. These algorithms [13] create a probabilistic generative model for the documents in the corpus. Each document is assumed to be a mixture of various topics. Each topic is characterized by a distribution over words and is analogous to a cluster. However, these systems create flat and fuzzy partitions where each document can belong to multiple clusters.

For all we know, the first application of conceptual clustering algorithms to documents was based on CLASSIT [14,15]. However, we identified a problem in that work. Despite proclaiming to be a fully incremental system, it uses a variant of TF-IDF to do feature selection at the very beginning. This procedure is usual in batch clustering, but cannot be done in incremental systems, since the documents to be processed are not yet known.

Other systems that claim to be incremental (e.g. Marcacini and Rezende [16]) start by an initial phase where a set of documents is clustered in a non-incremental way. Just then, in a second step, the upcoming documents are incrementally inserted in the initial hierarchy.

3. Proposed Approach

In this section, we describe the proposed approach and present the system we are developing. The goal of this research is the creation of OHDOCLUS, a document clustering algorithm that fulfills the following requirements:

- Online - Documents must be processed one at a time, and as fast as possible;
- Hierarchical - It must arrange the document collection in a hierarchy of clusters;
- Conceptual - Each cluster must have a probabilistic description of the documents it contains;

We must point out that our system is not distance-based, and this is one of the main differences to most document clustering algorithms. Since it does not need to compute the similarity between every pair of documents, it is scalable for huge datasets and is well suited for online processing. OHDOCLUS extends the COBWEB and CLASSIT algorithms to text data, implementing some adaptations to enable that extension. Although our system can accept any type of data, we use the term "document" to refer to each instance to be clustered.

A prototype of OHDOCLUS was implemented and has been used as a testbed for new options, validation, and comparison with other systems, and will serve as a benchmark for future versions.

The remaining of this section describes our system with more detail, focusing on document representation, dimensionality reduction, evaluation, the implemented algorithm, and also on the clustering evaluation.

3.1. Document Representation

Documents are submitted to the usual preprocessing steps (tokenization, case folding, stop words removal, and lemmatization) to reduce their size and improve the effectiveness of the algorithm. The current prototype accepts documents written in English or Portuguese, but any other language may be used if similar preprocessing tools are available.

Each document is represented by a vector following the *vector space model* [17]. Each value in this vector corresponds to a term occurring in the set of documents, and its value represents the importance of this term in the document.

For the time being, we have four possible options for the weights contained in the document vectors: term frequencies (TF), TF-IDF (Term Frequency–Inverse Document

Frequency) values, Boolean values (occurs/not occurs) and discrete ranges values (e.g. not occurs, rare, frequent). The last two schemes are nominal, being the result from the discretization of the TF or TF-IDF values, and can produce simpler clusters descriptions. In the case of TF and TF-IDF values, we implemented several of the many options and variants that have been suggested.

To deal with the problem of new words appearing in each incoming documents we use the *feature hashing* [18] (also known as *hashing trick*) – a mapping between terms and their indices. Instead of using a sequential hash table that needs to be updated with each new document, we apply a hash function to the features to determine their indices in the sparse document vectors.

Finally, we can use an additional vector of extra features (numerical, nominal or both) to each document. This vector is not submitted to any treatment and is just appended to the reduced document vector (resulting from document preprocessing and dimensionality reduction) to serve as input to the clustering algorithm. The additional vectors can be used for two purposes. First, to enrich the document representation with contextual information. Second, to enable the clustering of corpus previously processed and whose reduced document-term matrix was stored, without the need to reprocess the corpus. This last option can, ultimately, transform the system in a data clustering system that may be applied to non-textual data.

3.2. Dimensionality Reduction

Even after the preprocessing, the document vectors are high-dimensional and extremely sparse, which makes it difficult to detect relationships among terms and affects the performance and results. This problem, known as the *curse of dimensionality* [19], is an important issue in text mining.

Two types of dimensionality reduction techniques (DRT) have been proposed to overcome this problem [20], converting the initial high-dimensional full vectors in reduced vectors that will represent the document in the system:

- *Feature selection* [21] chooses a subset of the original features that is more informative than the whole set. The most used methods with text data sum the values of term frequencies or TF-IDF over all the documents and choose the terms with highest sums.
- *Feature transformation* creates a new space with lower dimension, where each dimension is a linear combination of the original features. Two of the most used feature transformation techniques are LSI (Latent Semantic Indexing) [22] and LDA (Latent Dirichlet Allocation) [23]. These techniques create an algebraic model that transform any vector into a new and more compact representation.

These DRTs are easy to use in data clustering, where the original features are predefined, and in batch document clustering, where the entire corpus is available from the outset. However, when we change to an online environment, dimensionality reduction becomes a challenging problem, and we are puzzled by the number of works that ignore it. Explaining in more detail, we have to deal with one of these problems:

- If we use feature selection, the incoming of a new document can change the set of selected terms, even if we use a local weighting scheme (that only depends on the content of the current document e.g. term frequencies); if we use global weights (such as the IDF component of TF-IDF), every document vector needs to be updated;

- If we use feature transformation, each new document forces the recalculation of the model and all the vectors previously computed. Even with some recent incremental versions of these methods, such as those implemented in Gensim [24], that enable online training to update the model incrementally with new documents, vector recalculation is always needed.

To deal with this issue, and being impossible recalculate everything from scratch with each document, we implemented a procedure of periodic updates with increasing interval between updates. During the update, the set of selected features or the model of feature transformations is recalculated, and every node in the hierarchy is updated accordingly. The interval between updates can be gradually increased since the impact of incoming documents becomes progressively smaller. Just in the very beginning, when the number of documents already processed is small, the changes in each update may be more significant. However, in that case, the tree is small, and the update is computationally cheap. When the number of documents in the hierarchy becomes significant, the changes in each update are more limited, and we expect that the impact in the hierarchy can be solved by the backtracking mechanism.

The documents received between updates are processed using the features or models computed in the last update, which can lead to some differences in the document vectors and in the hierarchy produced. However, we believe that these differences can be overcome in each update and the hierarchy reorganization that follows each update. Nonetheless, some experimentation is required to evaluate how the hierarchy is affected and if the final clustering is different from the clustering that would be produced if we perform an update after each document.

Other alternatives mentioned in the literature, like using statistics from an external and similar corpus to determine informative words in a "batch manner" or to compute alternatives to TF-IDF that are independent of other documents (e.g. TF-ICF [25]) seems infeasible or suffer from similar problems, as not avoiding the need of updates.

3.3. Evaluation Function

The search performed by OHDOCLUS in the space of cluster hierarchies is guided by an evaluation function based on cluster cohesiveness and cluster utility.

We define *cluster cohesiveness* as the measure of how closely related the documents placed in a cluster are. As in COBWEB, for each discrete feature F_i, the cohesiveness (COH) is given by the sum of the square of the conditional probability of each possible value of the feature:

$$COH_d(C,i) = \sum_j P\left(F_i{=}v_{ij}|C\right)^2 \quad (1)$$

For each numeric feature, CLASSIT defines the cohesiveness, though not using this term explicitly, as the inverse of the standard deviation of the feature values ($1/\sigma_i$). Because there is a problem when all the values of a feature are equal in a cluster ($\sigma_i = 0$), it introduces the *acuity*, a parameter that can be viewed as the minimum perceptible difference, and imposes a minimum to the standard deviation.

In OHDOCLUS, we limit the cohesiveness to the range [0,1] to enable the mixture of discrete and continuous features. To that end, we changed the cohesiveness of each continuous feature to:

$$COH_c(C,i) = \frac{a}{max(\sigma_i,a)} \tag{2}$$

where *a* is the *acuity* and σ_i is the standard deviation of the feature. In this way, when σ_i is below the acuity, the cohesiveness of the feature has the maximum value (one).

The cohesiveness of a cluster C is the average of the cohesiveness of the features:

$$COH(C) = \frac{\sum_{i=1}^{nd} COH_d(C,i) + \sum_{i=1}^{nc} COH_c(C,i)}{nd+nc} \tag{3}$$

where *nd* and *nc* are the number of discrete and continuous features. If we use different weights for each feature, the cohesiveness is the weighted average over every feature.

Because the initial experimentation showed that the cohesiveness has a huge sensibility to the variation of the acuity (usually very small values), in last prototype we implemented an alternative way to compute the cohesiveness. Considering that it is easier to find a suitable value for the maximum than for the minimum standard deviation, we replaced the acuity by a new parameter σ_{max} that is the maximum standard deviation to be considered. With this change, the cohesiveness of a continuous feature is:

$$COH_c(C,i) = max(0, 1 - \frac{\sigma_i}{\sigma_{max}}) \tag{4}$$

Finally, the search in the space of hierarchies performed by OHDOCLUS is guided by *cluster utility* (CU), a measure of the improvement caused by the cluster and its children. For a clustering C composed by a parent cluster C_p and its N children $\{C_1, .., C_N\}$, the cluster utility is the increase in cohesiveness afforded by the clustering, with respect to the parent cluster:

$$CU(C) = \frac{\sum_{k=1}^{N} P(C_k) COH(C_k) - COH(C_p)}{N} \tag{5}$$

The division by N intends to prefer clustering with fewer clusters. Otherwise, the best clustering would always be composed of singleton clusters.

3.4. Algorithm

In this section, we describe the operators and the backtracking mechanism used by the algorithm.

3.4.1. Operators and control

The control strategy of the current version of OHDOCLUS is similar to those in COBWEB and CLASSIT. To classify a document in a node it considers four operators:

- *Insert* the document in one of the children clusters;
- *Create* a new cluster for the document;
- *Merge* a pair of clusters into a new cluster (moving down the original clusters);
- *Split* a cluster (promoting its children).

For each operator, the algorithm evaluates the resulting clustering and applies the operator that yields the highest clustering utility.

The general control strategy of OHDOCLUS is:

```
C = root of the hierarchy
WHILE C is not a leaf
   Update the probabilistic concept in C
   Test Insert document in each child of C
   Test Create a new child of C
   Test Merge every pair of children of C
   Test Split each child of C
   Apply the operator with the best score
   IF backtracking operator was used
   THEN continue in updated C
   ELSE proceed to next level
```

With this strategy, the system tends to converge on cluster hierarchies in which the first level is the optimal partition (with respect to CU) over the entire corpus [9].

3.4.2. Backtracking

With only the first two operators, the system would be very sensitive to the order of documents presentation. The merge and split are two special operators that implement the *backtracking* mechanism. Being roughly inverse operators, merge and split enable the system to move in the space of hierarchies and adjust the number of clusters, thus enabling the system to recover from earlier decisions that later prove to be wrong. When one of these operators is used, the classification of the document is "halted" to rearrange the hierarchy locally. After this, the document classification proceeds in the same point.

We intend to improve this backtracking mechanism to have, at each moment, a tree that reflects as much as possible, the documents processed so far. With this aim, we have extended the scope of the original operators, bearing in mind that a better hierarchy (even at the cost of some additional time) can prevent future and more expensive corrections. While the original merge operator considers only the two best children of the node, our prototype can consider all possible pairs of children. Similarly, the current split operator considers every child while the original tests only the best cluster. We are assessing several possibilities to find the best scope regarding the tradeoff between processing time and clustering quality,

We are also considering the implementation of new operators (for example, node promotion) and detaching the backtracking operators from the other two operators. This option would enable a faster document insertion (even if not in the optimal place) but would be followed by backtracking operations to improve the quality of the hierarchy. We could even think in a process of iterative optimization or cluster refinement running permanently in parallel.

3.5. Document storage

The original algorithm used a *cutoff* parameter to stop the classification when the utility is too low. When this occurs, the document is "hidden" in the node to simplify the hierarchy. In a preliminary version of OHDOCLUS, we kept this parameter. However, the experimentation showed that the existence of hidden documents in non-leaf nodes of the tree could be meaningless in our system, due to the frequent updates. In the current version, we removed this parameter and introduced an optional parameter with the maximum height of the tree. If this parameter is not defined, there is no limit to the number of levels in the hierarchy. When the user defines a maximum height, the documents are stored in the node when they reach the maximum level.

We have to maintain all the documents in the tree because we need to recalculate the document vectors and the clusters' statistics every time we make an update. This recalculation cannot be considered a document's reprocessing, which would violate one of the assumptions of online algorithms because we just update the vector that represents it in the hierarchy and, eventually, reorganize the tree.

3.6. Clustering Evaluation

The task of evaluating the quality of a clustering is "the most difficult and frustrating part of cluster analysis" [2]. Since our algorithm does not use the distance between documents, we cannot use most usual measures of internal clustering quality. Hence, we use cluster cohesiveness and average clustering utility as internal measures of quality (without respect to any external information).

With categorized corpora, where the documents have a preassigned class label, we can use external indexes to assess how the clustering matches the supplied ground truth. In a first moment, we used the traditional precision, recall, and F-measure [26]. However, as there is no predefined number of clusters, we were faced with the limitations of those metrics. For instance, precision is biased towards small clusters, reaching a maximum when all clusters are of size one. Also, these metrics just consider the most common category in each cluster, thus giving the same evaluation to different clusterings.

As none of the metrics we are using satisfy all the four criteria for an effective external measure [27], we decided to implement the BCubed measures, which combine the best features from other metric families and are becoming increasingly popular in clustering evaluation [28]. We use *BCubed F-Score,* a measure that combines BCubed precision and recall to satisfy all the criteria [29]. They consider the overall disorder of each cluster and not just the predominant category. Nevertheless, clustering evaluation remains a tricky matter, even because "clusters are in the eye of the beholder" [30].

4. Preliminary Experiments

We build a prototype of our system to do some experimentation and validation. We present here some preliminary results and some conclusions that can be drawn.

For an initial validation, we built a small ad-hoc corpus of 20 short news (about 200 words each), written in two languages (10 in Portuguese and 10 in English). We used it with several setups and verified that the system can build the optimal partition at the first level in about 90% of the experiments, identifying the two obvious clusters (Figure 1), even with a vocabulary of just 5 terms. The system does not find the optimal partition when the 10 documents of each language are presented consecutively. This is the only case where the backtracking operators cannot fix the hierarchy, and many clusters are created in the first level.

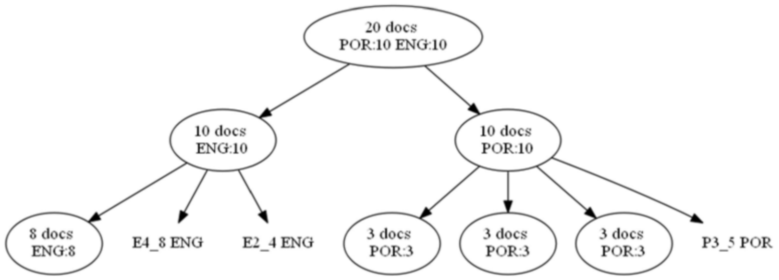

Figure 1. First levels of the clustering produced with 20 documents in Portuguese and English

In the next experiences, we used the *Brown*[2] corpus, a selection of 500 published texts in English, categorized in 15 genres. We wanted to compare the quality of the partitions produced by our system at the first level of the hierarchy with those obtained with a batch k-means algorithm. This algorithm receives the number of categories and the final document vectors and processes them as a whole. We present in Table 1 the results of some experiences with subsets of the Brown corpus, composed by documents drawn from the categories with more documents.

Table 1. Results from experiences with Brown corpus with OHDOCLUS and batch K-Means

Categories	Documents	Weights	Terms	Clusters	OHDOCLUS BC F-Score	K-means BC F-Score
2 (news, belles_lettres)	50	TF	50	3	0.667	0.538
			100	2	0.926	0.538
			200	2	0.855	0.538
		TF-IDF	50	2	0.515	0.667
			100	2	0.888	0.667
			200	2	0.961	0.667
3 (news, belles_lettres, lore)	120	TF	50	3	0.462	0.433
			100	2	0.437	0.471
			200	2	0.498	0.404
		TF-IDF	50	3	0.390	0.508
			100	2	0.425	0.508
			200	2	0.530	0.486

These results show that the algorithm can separate, almost always, the documents from the 2 categories and performs better that K-means in that case. This can be explained by the Euclidean distance used by K-means, that is not the best to use with documents. With more than 2 categories OHDOCLUS has more difficulties to find the right clusters and the results are somewhat erratic. Much more experimentation is needed to understand the results and to find the best values for the parameters for each case.

We also run some experiments to confirm that backtracking operators are really needed. We turned off the backtracking operators and repeat the experiments of Table 1 with 2 categories and we observed that the clusterings obtained have very low quality (BCubed F-Scores between 0.504 and 0.554), with many clusters (between 3 and 6) in the first level.

Next, we tried to figure out how many documents are needed until the set of selected features stabilize. We verified that after an initial phase of many changes (first 10 documents), the set of selected features stabilizes gradually, and the changes become

[2] http://www.nltk.org/nltk_data/

more and more rare. These results support our idea of more frequent updates in the start (when there are more changes but the updates are faster) and progressively increase the intervals between updates.

Finally, we used the classical 20 Newsgroups corpus, containing about 20000 posts, and we observed that the algorithm scales well, since the document processing time increases only slightly with the number of documents.

From these preliminary experiments, we conclude that further work is needed to gain a deeper insight into the system behavior.

5. Conclusions and Further Work

In this paper, we have proposed OHDOCLUS, a document clustering algorithm that combines online and hierarchical clustering. Unlike most document clustering algorithms, OHDOCLUS is not based on the distance between documents, but on the clustering utility.

This research is an ongoing work and, so far, only a prototype was developed for testing the proposed approach. In a near future, we aim to implement and test additional features, at different levels, that will, hopefully, improve the system performance.

Concerning document representation and dimensionality reduction, we intend to:

- Compare and evaluate the clusterings obtained using different weighting schemes and different dimensionality reduction techniques;
- Evaluate if the additional features that can be attached to a document are useful;
- Analyze the sensitivity to different sizes of the set of selected features and try to define an appropriate value for each corpus automatically;
- Evaluate how the tree is affected by the changes in the set of selected terms;
- As some dimensionality reduction methods require a significant number of documents to produce useful results, we will test the use of one method in the beginning and later, when the corpus is large enough, change to another method.

With respect to the operators and the control strategy, some possible paths are:

- Isolate the document classification from the tree reorganization;
- Consider the implementation of other operators to refine the hierarchy;
- Assess and validate the utilization of σ_{max} and find the best value for it.

To determine how the online requirement degrades clustering quality, we will:

- Compare the clusterings obtained with those created by batch systems;
- Study how the order of document presentation affects the clusterings produced.

We do believe that our work can contribute to solve some open issues in online hierarchical document clustering and to create a framework for future research in this field.

References

[1] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, 9th ed. John Wiley & Sons, 2005.

[2] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[3] M. Steinbach, G. Karypis, and V. Kumar, “A Comparison of Document Clustering Techniques,” Technical Report Technical Report #00-034, 2000.

[4] C. C. Aggarwal and C. Zhai, “A Survey of Text Clustering Algorithms,” in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Boston, MA: Springer, 2012, pp. 77–128.

[5] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, Jun. 2010.

[6] B. C. M. Fung, K. Wang, and M. Ester, “Hierarchical Document Clustering,” in *Encyclopedia of Data Warehousing and Mining*, 2nd ed., IGI Global, 2009, pp. 970–975.

[7] M. Steinbach, G. Karypis, and V. Kumar, “A Comparison of Document Clustering Techniques,” in *KDD Workshop on Text Mining*, 2000, vol. 400.

[8] D. H. Fisher, “Conceptual Clustering,” in *Handbook of Data Mining and Knowledge Discovery*, W. Klosgen and J. Zytkow, Eds. Oxford University Press, 2002, pp. 388–396.

[9] D. H. Fisher, “Knowledge Acquisition Via Incremental Conceptual Clustering,” *Mach. Learn.*, vol. 2, no. 2, pp. 139–172, Sep. 1987.

[10] J. H. Gennari, P. Langley, and D. H. Fisher, “Models of Incremental Concept Formation,” *Artif. Intell.*, vol. 40, no. 1–3, pp. 11–61, 1989.

[11] M. A. Gluck and J. E. Corter, “Information, uncertainty and the utility of categories,” in *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, 1985, pp. 283–287.

[12] Q. He, K. Chang, E.-P. Lim, and J. Zhang, “Bursty Feature Representation for Clustering Text Streams,” in *Proceedings of the SIAM Conference on Data Mining*, 2007, pp. 26–28.

[13] D. M. Blei, “Introduction to Probabilistic Topic Models,” *Commun. ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[14] N. Sahoo, “Incremental Hierarchical Clustering of Text Documents,” Carnegie Mellon University, New York, New York, USA, 2006.

[15] N. Sahoo, J. Callan, R. Krishnan, G. Duncan, and R. Padman, “Incremental Hierarchical Clustering of Text Documents,” in *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM’06)*, 2006, pp. 357–366.

[16] R. M. Marcacini and S. O. Rezende, “Incremental hierarchical text clustering with privileged information,” in *Proceedings of the 2013 ACM Symposium on Document engineering - DocEng ’13*, 2013, pp. 231–232.

[17] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975.

[18] K. Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, “Feature Hashing for Large Scale Multitask Learning,” in *Proceedings of the 26th International Conference on Machine Learning (ICML’09)*, 2009.

[19] A. Zimek, “Clustering High-Dimensional Data,” in *Data Clustering: Algorithms and Applications*, C. C. Aggarwal and C. K. Reddy, Eds. CRC Press, 2013, pp. 201–230.

[20] B. Tang, M. Shepherd, E. E. Milios, and M. I. Heywood, “Comparing and Combining Dimension Reduction Techniques for Efficient Text Clustering,” in *Proc. of the International Workshop on Feature Selection for Data Mining*, 2005, pp. 17–26.

[21] S. Alelyani, J. Tang, and H. Liu, “Feature Selection for Clustering: A Review,” in *Data Clustering: Algorithms and Applications*, C. C. Aggarwal and Chandan K. Reddy, Eds. CRC Press, 2013, pp. 29–60.

[22] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by Latent Semantic Analysis,” *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, Sep. 1990.

[23] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” *J. Mach. Learn. Res.*, vol. 3, no. 4–5, pp. 993–1022, 2003.

[24] R. Rehurek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010, pp. 45–50.

[25] J. W. Reed, Y. Jiao, T. Potok, B. A. Klump, M. T. Elmore, and A. R. Hurson, “TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams,” in *Proceedings of the 5th International Conference on Machine Learning and Applications (ICMLA’06)*, 2006, pp. 258–263.

[26] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[27] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, “A Comparison of Extrinsic Clustering Evaluation Metrics based on Formal Constraints,” *Inf. Retr. Boston.*, vol. 12, no. 4, pp. 461–486, 2009.

[28] E. Amigó, J. Gonzalo, and F. Verdejo, “A General Evaluation Measure for Document Organization Tasks,” *Proc. 36th Annu. Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, p. 643, 2013.

[29] A. Bagga and B. Baldwin, “Entity-Based Cross-Document coreferencing using the Vector Space Model,” in *Proceedings of the 36th annual meeting on Association for Computational Linguistics*, 1998, vol. 1, pp. 79–85.

[30] V. Estivill-Castro, “Why so many clustering algorithms - A Position Paper,” *ACM SIGKDD Explor. Newsl.*, vol. 4, no. 1, pp. 65–75, 2002.