Incremental and Hierarchical Document Clustering

Rui Encarnação and Paulo Gomes

CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
{race, pgomes}@dei.uc.pt

Abstract. Over the past few decades, the volume of existing text data increased exponentially. Automatic tools to organize these huge collections of documents are becoming unprecedentedly important. Document clustering is important for organizing automatically documents into clusters. Most of the clustering algorithms process document collections as a whole; however, it is important to process these documents dynamically. This research aims to develop an incremental algorithm of hierarchical document clustering where each document is processed as soon as it is available. The algorithm is based on two well-known data clustering algorithms (COBWEB and CLASSIT), which create hierarchies of probabilistic concepts, and seldom have been applied to text data. The main contribution of this research is a new framework for incremental document clustering, based on extended versions of these algorithms in conjunction with a set of traditional techniques, modified to work in incremental environments.

Keywords: conceptual clustering, dimensionality reduction, document clustering, hierarchical clustering, incremental clustering, vector space model

1 Introduction

There is a growing gap between the rate of generation of documents and our ability to organize and use them. This overload of information demands automatic tools to organize documents. Document clustering [1, 3] arranges documents into automatically created clusters.

Clustering is the division of data into clusters or "the art of finding groups in data" [12]. Each group (cluster) is made of objects that are similar between themselves (high intra-cluster similarity) but dissimilar to objects of other clusters (low inter-cluster similarity). Unlike classification, where a set of predefined classes is provided, in clustering the system must decide not only to which cluster each object must be assigned but also which clusters must be created. The items to be clustered are usually represented by a set of features, denoted by a vector of numeric or nominal values.

The main difference between document clustering and the general data clustering is the non-existence of a predefined set of features, since the features are defined from the content of the documents.

When dealing with documents the most used representation is the vector space model [19] where each document is represented by a vector of numeric features. Usually the features are the words occurring in the collection and we call it the bag-of-words model

because a document is represented as a set of words. A collection of *n* documents containing *m* different terms, can be represented by a term-document matrix of dimensions *m* x *n*. This matrix is extremely sparse because a document contains only a few hundreds from the tens of thousands of different words contained in the collection. There are many different ways to determine the weight of each word [13]. The most obvious is term frequency (*tf*) that counts how many times the term appears in the text. The simplest is term occurrence, a binary value indicating if the term occurs in the text. However, the most used weighting scheme in document clustering is TF-IDF that combines term frequency and inverse document frequency to assign a weight to each term in the document [20]. The most usual formula for TF-IDF is:

$$tf\text{-}idf_{d,t} = tf_{d,t} \times idf_t = tf_{d,t} \times log\frac{N}{df_t} \quad (1)$$

where N is the total number of documents and df_t is the document frequency or the number of documents that contain the term *t*. Thus, the weight assigned by TF-IDF to a term in a document is high when the term occurs frequently in a small number of documents (giving a high discriminating power) and low when the term occurs in many documents or occurs fewer times in a document.

The use of the vector space model to represent documents leads to a huge number of features. Usually, there are some thousands of terms in a vocabulary, making the term space high-dimensional. This is the *curse of dimensionality*, the main issue of text clustering. High dimensionality leads to very sparse vectors and makes it harder to detect the relationships among terms. High dimensionality also increases time and space complexity, affecting dramatically the performance. To overcome these problems, two types of dimensionality reduction techniques have been proposed [22]:

- Feature transformation, which projects the original high dimensional space onto a lower dimensional space where each new dimension is a combination of the original features, widely used examples being LSI [4] and LDA [9];
- Feature selection, which selects a subset from the original features, based on a measure computed from the document collection.

Many clustering algorithms [10] have been proposed but most of them cannot be applied to documents, because text data poses several new challenges:

- Scalability - Many clustering algorithms perform well on small datasets, but some fail to handle real world datasets containing millions of documents;
- Dimensionality - The number of terms can reach tens of thousands;
- Sparsity - The vast majority of entries in the term-document matrix is zero;
- Word correlation - The number of concepts is much smaller than the feature space;
- Cluster descriptions - The hierarchy created by the algorithm should contain meaningful cluster descriptions, to enable interactive browsing;
- Prior domain knowledge - Many clustering algorithms require some input parameters (the number of clusters being the most common one). Often, the user does not have such knowledge or makes wrong choices, so the clustering quality may be poor.

Document clustering has been used with many purposes: to organize the results returned by a search engine (creating a more effective presentation to the user), to improve information retrieval systems, to enable browsing collections of documents, to

find the nearest neighbors of a document, to detect new topics in news streams, and even to ascribe authorship of texts.

Document clustering algorithms can be classified in many ways [1, 3, 10]. According to the number of levels of the clustering created, the methods can be divided in hierarchical clustering which creates a tree of clusters, or partitioning clustering which creates a flat partition of clusters. According to the way the algorithm processes the documents, they can be classified in batch algorithms if the document set is processed as a whole (possibly with several iterations), and incremental algorithms, which process one document at a time, without the need of reprocessing the previous documents. Another division can be made between deterministic and probabilistic algorithms. Deterministic algorithms assign each document to only one cluster, while in probabilistic (or fuzzy) algorithms a document can be assigned to several clusters, possibly with different probabilities (soft clustering).

In our proposed approach, we choose hierarchical clustering because the hierarchy of clusters enables a quick browse of the topics of interest, allowing for searches at different levels, and it circumvents the problem of specifying the number of clusters, producing a satisfactory solution for users with different needs.

Most existing systems require that all the documents be present at the start, but this restriction is hard to satisfy in cases where documents are constantly being added. Clustering algorithms should be able to process a text as soon as it arrives, without reprocessing the previous ones, but maintaining the hierarchy updated. Such a situation calls for an incremental approach, which we believe will become more and more important.

In addition, most approaches to document clustering require a similarity measure and rely on distance between documents. This makes hierarchical clustering hard to apply to text data because it can be computationally expensive. We propose the use of conceptual clustering, an approach mostly used with non-textual data.

The main contributions of this work will be:

- An incremental and hierarchical algorithm based on COBWEB [5] and CLASSIT [7];
- The definition of a new framework for incremental document clustering, with the adaptation of some traditional techniques;
- The development of more sophisticated mechanisms of backtracking and tree reorganization which reduce sensitivity to the order of the documents;

In the next section, we present the most important work in this field. In section 3, we will describe our approach and the options made in our system. Next, we present some preliminary experiments and we will end with some ideas for future research.

2 Related Work

For the time being, we will review the clustering algorithms related to our work. We start with a brief description of the two main types (hierarchical and partitional) to describe later the two algorithms in which our system is based on. We will end with a brief overview of other incremental systems.

2.1 Hierarchical Clustering

Hierarchical clustering [6] organizes clusters into a tree, with the root containing the entire corpus of documents and leaves containing single documents. Depending on the direction of tree construction, hierarchical methods can be classified as divisive (starting with a node with all the documents and iteratively splits it until only singleton nodes remain) or agglomerative (starting with a node for each document and joining the most similar pair of clusters, in a bottom-up fashion, until the most general node is created). Unlike other techniques, here we do not need to specify the number of clusters.

Hierarchical methods usually suffer from their inability to recover from a wrong decision (a merge or a split). Furthermore, due to the complexity of computing the similarity between every pair of clusters, these methods are not scalable for handling large datasets, since the time complexity is quadratic with the number of documents.

2.2 Partitional Clustering

K-means [21] is the most widely used clustering algorithm due to its simplicity and efficiency. It is a distance-based algorithm, which builds a flat partition of *K* clusters. This number is a user-predefined parameter. K-means is based on the idea that a center point (centroid) can represent a cluster. The algorithm starts with a random set of *K* initial centroids and assigns each object to its closest centroid. Then, iteratively, new centroids are calculated from the objects assigned to each cluster and assignments are changed if necessary. The algorithm finishes when no reassignment occurs.

The run time of K-means and its variants is very attractive when compared to hierarchical techniques, particularly if the dataset is large. One disadvantage of K-means is the need to specify the number of clusters, which may lead to poor clustering. The algorithm is too sensitive to the initial seeds and can find itself trapped in a local minimum. Noise and outliers can also be a problem, as a small number of points can influence the mean value.

2.3 Conceptual Clustering

Conceptual clustering algorithms are incremental methods and build a hierarchy of probabilistic concepts. COBWEB and CLASSIT are the most notable among them.

COBWEB [5] is an algorithm that clusters objects described by categorical values. It creates a hierarchy where each node stores the conditional probability of each possible value for every attribute (probabilistic concept). The algorithm traverses the tree top-down until it reaches a leaf, updating the values in the nodes.

It is not a distance-based method. Instead, COBWEB uses *category utility* [8] as the evaluation function. Category utility is a heuristic trade-off between intra-class similarity and inter-class dissimilarity. Intra-class similarity ($P(A_i=V_{ij}|C_k)$) measures the *predictability* of the class for that value of the attribute. Likewise, the inter-class dissimilarity ($P(C_k|A_i=V_{ij})$) measures the *predictiveness* of the class given the value of the attribute. A clustering is good if features are predictable by the classes and, simultaneously, predictive of classes, and this should be more important for values that are more

common. This trade-off can be expressed as the product of predictability, predictiveness and commonness and - using Bayes' Rule - we get $P(C_k)\sum_i\sum_j P(A_i{=}V_{ij}|C_k)^2$. The inner expression is the expected number of values one could guess, given the class and following a *probability matching strategy*, where a value is guessed with the same probability of its occurrence. Finally, the category utility of a clustering $C = \{C_1, .., C_N\}$ is defined as the average of the category utility of the clusters:

$$CU(C) = \frac{\sum_{k=1}^{N} P(C_k)\left[\sum_i\sum_j P(A_i{=}V_{ij}|C_k)^2 - P(A_i{=}V_{ij})^2\right]}{N} \quad (2)$$

When classifying an object in a node, the algorithm considers four possible operators:

- classify the object in an existing cluster;
- create a new cluster with the object;
- merge the two clusters with the highest scores into a new cluster;
- split the cluster with the highest score in its children;

Finally, it performs the one that yields the highest CU value.

CLASSIT [7] is the successor of COBWEB and overcomes some of its limitations. It uses only numeric attributes and this implies a generalization of the evaluation function used in COBWEB because the domain is continuous. Since there is no prior information about the distribution of the values, the best estimation is the normal (Gaussian) distribution. With this distribution, the squared probabilities in CU formula are replaced by $1/\sigma_i$ where σ_i is the standard deviation of the attribute *i*. There is a problem with this expression when standard deviation is zero (all objects in a cluster have the same value for an attribute). To solve this, CLASSIT uses *acuity*, a parameter that imposes a minimum standard deviation. Another useful parameter used in CLASSIT is *cutoff*, which enables stopping the classification whenever the category utility drops below that threshold. This prevents the system from continuing until it reaches a leaf, increasing performance, and reducing hierarchy complexity (as well as the risk of overfitting).

We can say that COBWEB and CLASSIT do a hill-climbing search in the space of concept hierarchies guided by category utility. The merge and split operators enable recovering from previous wrong decisions and make the algorithms less sensitive to the order of instances. These systems satisfy the criteria proposed for incremental clustering evaluation [5]: the cost of incorporating an instance is low, the hierarchies produced have high quality and promote correct predictions. They introduce some interesting features in clustering such as an evaluation function grounded in Probability Theory, the backtracking mechanisms, and the use of conceptual clustering for inference.

2.4 Incremental Document Clustering

Some research in incremental text clustering has been done in the field of topic modeling to detect and track new events in a stream of online news [2]. Topic modeling uses a time stamp attached to each text and a decaying function to reduce the importance of older texts. Generally, these systems create flat partitions where each document can belong to more than one cluster simultaneously (fuzzy clustering).

As far as we know, the first utilization of conceptual clustering algorithms in document clustering was [16, 17]. That work was based on a modified version of CLASSIT. It stated that the normal CLASSIT, without modifications, is unsuitable for document clustering because it assumes that the values of the attributes follow a normal distribution. This is a valid assumption for generic attributes, but not for word occurrences in documents, which are non-negative integer counts. The frequency of words in a document collection follows Zipf's law [24] - the frequency of any word is inversely proportional to its rank in the frequency table, sorted in decreasing order.

The authors presented several possibilities to model the occurrence of words across documents and selected the Katz's K-mixture model [11] since it is simpler and models the occurrences of words in the documents better than most other distributions, and about as well as the Negative Binomial (which fits the word occurrence very well, but is computationally expensive). Some experiments to assess the quality of the clustering obtained with Katz's distribution versus the Normal version are presented. Katz's version was expected to be much better, but the results were unclear. The tests also showed that Katz's version is slower than Normal version.

We identified two issues in this often-cited work:

They use TF-IDF to calculate the document vectors but when trying to model the distribution of values they consider just term frequency. This is a problem because TF-IDF values are not integer counts and Zipf's law cannot be applied to TF-IDF. Thus, there is no clear proof that Katz's distribution is a fair assumption for TF-IDF values.

A more serious issue is the utilization of a variant of TF-IDF in dynamic feature selection. This is a usual choice in batch clustering, but cannot be used in an incremental system without modifications. As the documents are not present at the beginning, we cannot use global measures as IDF. We can use traditional datasets to establish comparisons with other systems, but simulating documents will arrive one at a time. In a subsequent version of the work [18], all references to feature selection were omitted. We contacted the main author and he agreed with our point of view.

3 Proposed Approach

In this section, we will describe the approach followed in this research, explain the options made, and show the current stage of development of our system.

The main goal of this research is the creation of a new document clustering algorithm with a specific set of requirements:

- It must create a hierarchy of clusters of documents - *hierarchical*;
- Each node must have a probabilistic description of the documents it contains - *conceptual clustering*;
- Documents are processed one at a time, with a processing rate greater than the expected arrival rate - *incremental*;
- No additional information should be required - *unsupervised*.

We want to emphasize the main differences between our system and those earlier described. Our system does not rely on the distance between documents, since the document classification is guided by category utility; this makes it more scalable and enables

the execution of tree reorganization in parallel with document classification. Although it is based on a combination of COBWEB and CLASSIT, it is specially designed for incremental document processing, extending the previous operators, and implementing new incremental techniques for dimensionality reduction.

We have implemented a preliminary version of our HIDOCLUS algorithm. This early implementation is used for comparison against other published results, and serves as a benchmark for future versions and as a test bed for experimentation on document representation.

The algorithm was implemented in Python, using NLTK[1] - a widely used framework for NLP that provides access to traditional corpora and text processing libraries - and Gensim[2] - a Python library designed to efficient processing of large corpora, with incremental implementations of many algorithms (including TF-IDF, LSI, and LDA) [15] which do not require documents to stay in memory.

3.1 Document Representation

HIDOCLUS accepts documents in English or Portuguese (any other language can be used, provided the necessary preprocessing tools are available) and performs some standard preprocessing steps to reduce the size of the representation and improve the effectiveness of the algorithm [13]:

- Tokenization – splits the document into individual words (tokens)
- Normalization – converts all characters to lower-case (case folding);
- Stopwords removal – removes all the tokens contained in a predefined list of frequent words with little discriminative power (articles, prepositions, conjunctions);
- Lemmatization – ignores the inflected forms of a word and retains only the lemma.

For the moment, we use vector space model with numerical attributes and TF-IDF weights. However, the system is prepared for two additional possibilities.

First, we can enrich the document representation with extra attributes providing contextual or additional information. Since our system accepts a mix of both types, these attributes can be numeric or nominal and they can have an associated weight. (However, the presence of nominal attributes can be incompatible with some of the feature transformation techniques for dimensionality reduction.)

Another possibility is the transformation of term frequencies (or even TF-IDF values) in nominal attributes ("discretization"). This model will be faster, produce simpler cluster descriptions, and overcome the problem of model the frequencies distribution.

3.2 Dimensionality Reduction

Many document clustering batch algorithms use TF-IDF to select the most informative terms. However, to achieve it they need to know a priori the document frequency of

[1] http://nltk.org
[2] http://radimrehurek.com/gensim

every term, which is impossible in incremental systems, since we do not have a previous knowledge of the corpus. This is a key challenge in incremental document clustering systems and, strangely, has been ignored in many works in this field, as in [17] and many subsequent works that used it acritically. To overcome this problem, some solutions are possible:

- Use incremental versions of the dimensionality reduction techniques that transform the set of initial feature in a much more reduced set, e.g. LSI or LDA.
- Use a weighting scheme not affected by new documents like TF-ICF [14]. This scheme replaces IDF by inverse corpus frequency (ICF) of other known corpus. This idea is similar to the use of training data in classification. TF-ICF is fast and effective in general domains if a large representative corpus is available. However, in specific domains, an appropriate corpus can be harder to determine;
- Use TF-IDF but recalculate periodically the selected features and the document vectors. This recalculation updates the system with the information conveyed by the documents that have arrived in the meantime, since the IDF of previous documents is affected by each incoming document. The interval between recalculations can be progressively increased, since the impact of adding a new document is becoming gradually smaller.

The current version of HIDOCLUS uses the latter. Whenever we need to update the set of K most informative features, we compute the sum of TF-IDF for each term, across all the documents so far processed, and choose the K with highest scores. In this stage, there is a fixed interval between "refreshes" but we intend to increase dynamically the interval in the future.

Our initial work plan did not include the utilization of LSI and related semantic techniques for dimensionality reduction, since the recalculation of the model from scratch - with the arrival of each document - will forbid its use in incremental environments. However, having found the incremental versions of these algorithms used by Gensim, we have already implemented LSI and are actually working on LDA implementation.

3.3 Evaluation Function

The evaluation function used by HiDoclus is based on the functions used by Cobweb and Classit. When the dataset contains attributes of only one type, the function is essentially the same used by these systems.

In the case of numerical attributes, for the moment, and in the absence of a clear better solution, we decided to maintain the assumption of CLASSIT and the evaluation is based on the standard deviation of the attributes in each cluster:

$$CU(C) = \frac{\sum_{k=1}^{N} P(C_k)\left(\sum_i \frac{1}{\sigma_{ik}} - \frac{1}{\sigma_{ip}}\right)}{N} \quad (3)$$

where C is a cluster with N children, σ_{ik} and σ_{ip} are the standard deviation of the attribute i, in child cluster k and in parent cluster p, respectively. We maintain the *acuity* parameter to specify the minimum value for σ and prevent clusters with null deviation.

When mixed type attributes are used, the function is the sum of (2) and (3).

3.4 Algorithm

The control strategy and learning mechanism used in the preliminary version of the HIDOCLUS algorithm are very similar to those used in COBWEB and CLASSIT.

When inserting a document in a hierarchy of clusters we use the same four operators:

- Place the document in an existing cluster (it tries every child of the current cluster);
- Create a new cluster with the incoming document;
- Merge two clusters into a new cluster (we extend this operator to consider the merge of every pair of children and not only the two with highest scores);
- Split a cluster, promoting its children (we have also extended this operator to consider every child of the current cluster and not only the best);

These two last operators implement the backtracking mechanism to recover from previous wrong choices. Whenever the algorithm uses these operators, it stops the classification of the document for a while to rearrange locally the hierarchy. After this, the classification proceeds in the same node. We present the algorithm below:

```
N = root of the tree
WHILE N is not a leaf
  Update the probability concept of N
  Compute the score of placing Document in each child of N
  Compute the score of creating a new child of N with Document
  Compute the scores for using the Merge and Split operators
  IF best score < cutoff
  THEN Stop classification
  ELSE
    Apply the operator with the best score
    IF Merge or Split are used
    THEN Proceed with the classification in updated Node
    ELSE Proceed to next level (best or new child of Node)
```

4 Experimentation

Since our research is yet in an early stage, much work is still in progress, and many implementation decisions need to be taken and validated. However, some preliminary experiments have been conducted and a number of conclusions can already be drawn.

First experiments with non-textual data allowed us to validate the algorithm and confirm that it produces the same results as COBWEB and CLASSIT, for the same dataset.

Next, we used reduced versions of several NLTK corpora to study the sensitivity of the algorithm to its parameters. The cutoff parameter stops the classification whenever the category utility drops below that threshold. This prevents the system from continuing until it reaches a leaf, reducing the height and complexity of the hierarchy. A null cutoff forces the system to retain all documents as leaves of the tree, leading to an overfitting effect, as the algorithm will sort new documents as far down as possible. With higher values, the system produces simpler hierarchies with more general clusters as terminal nodes. This reduces notoriously the processing time of future documents.

The acuity parameter can be thought as the minimum perceptible difference, setting the breadth of the hierarchy. Low values of this parameter increase the branching factor of the tree and increase the number of singleton clusters. However, the role of this parameter is not clear, since it also affects the category utility of clusters having attributes with null or very small deviation. In this case, the value of CU is given by the product of the inverse of acuity by the number of features used. This is usually a big value since the number of features is large - even after dimensionality reduction – and the acuity is an extremely small value (same order of magnitude of the weights in document vectors, due to normalization). In order to get a deeper understanding of this relation between acuity, number of features and CU values, we need to do additional experiments.

We also evaluated the behavior of merge and split operators, using several possibilities for the nodes considered by the operators. We verified that processing time is very similar in all the cases (less than 10% between the best and worst cases). Testing more nodes takes longer but the system makes better choices, which saves time in future steps. We noted a slight increase in clustering quality when the split operator considers all the nodes. The merge operator obtains best results when it considers only the two best nodes, but the difference is insignificant. For now, we keep considering every node, until further experiments enable a deeper insight.

In other experiments, we used tagged corpora from NLTK (Brown and Movie Reviews) to compare the top levels of the obtained hierarchy with the predefined classes. The results ranged from 50% to 85% of purity in clusters at top levels. We think this is caused by the variability of category utility values aforementioned.

We have also studied the distribution of TF-IDF values, and we observed that the vast majority of values are zero or near zero. This is caused by the sparsity of the space and the normalization process.

Finally, we tried to figure out how many documents are needed to vocabulary stabilization. We ran experiments where the vocabulary is recalculated from scratch after the arrival of every new document. We observed an expected initial period (5 to 10 documents) of big changes, but then the vocabulary stabilizes gradually and the changes become more and more rare. For instance, with a corpus of 1000 documents from the Reuters corpus and a vocabulary size of 50, we see that after the 20^{th} document none caused more than two changes, with the majority causing no change. This reinforces our belief in increasing intervals between refreshes.

5 Future Work

Many initial ideas are still not implemented and many other possible paths are suggested by the work done so far. Among them, we highlight the following:

- Implement feature hashing (aka hashing-trick) [23], an extension of VSM that uses a hash dictionary. This approach speeds-up the model, leads to huge savings in memory, and allows adding new features dynamically;
- Compare the hierarchies obtained using different weighting schemes and different dimensionality reduction techniques;

- Determine an appropriate number of features for each corpus and study the sensitivity of the algorithm to different values for the number of features. Additional experimentation is needed to study how the tree is affected by vocabulary changes;
- Standardize the range of values returned by the evaluation function;
- Find appropriate values for acuity and cutoff parameters for each corpus;
- Isolate the tree reorganization from the document processing. The hierarchy reorganization can be performed concurrently, in background, searching for outliers or portions of the tree with low quality. The merge and split operators could be isolated from the other two operators, and extended to perform more global operations;
- Allow a document to be classified in more than one cluster (fuzzy clustering);
- Implement other cluster quality measures (internal and external) for more accurate evaluation of the produced hierarchies;
- Study to what extent the incremental requirement degrades clustering quality, comparing the results obtained with different orders of document presentation and then compare the clustering produced by our system with the results produced by batch systems knowing the whole corpus at the beginning.

6 Conclusion

This research is an answer to the growing demand for automatic document organization tools. We present a new document clustering application combining two features seldom used in conjunction: incremental and hierarchical. To make this possible, we adapt some of the usual techniques in Text Mining to an incremental environment. At the same time, we seek the best solutions for some open issues in incremental documental clustering such as the document representation and the dimensionality.

We believe that our work could be important to create a framework for future incremental and hierarchical document clustering research and we think our system to be an invaluable tool to prevent us from being overwhelmed with documents.

Acknowledgements

This paper was submitted to the SDIA 2013 - 4th Doctoral Symposium on Artificial Intelligence and reports research work for the PhD Thesis of the Doctoral Program in Information Sciences and Technologies of Faculty of Science and Technology of the University of Coimbra. The research, which started in April 2011, is being supervised by Professor Paulo Gomes, and the foreseen conclusion date is September 2014. It is partially supported by the FCT scholarship grant SFRH/BD/73543/2010.

References

1. Aggarwal, C.C., Zhai, C.: A Survey of Text Clustering Algorithms. In: Aggarwal, C.C. and Zhai, C. (eds.) Mining Text Data. pp. 77–128 Springer, Boston, MA (2012).

2. Allan, J. et al.: On-line New Event Detection and Tracking. Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98). pp. 37–45 ACM Press, Melbourne, Australia (1998).
3. Andrews, N.O., Fox, E.A.: Recent Developments in Document Clustering. (2007).
4. Deerwester, S. et al.: Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science. 41, 6, 391–407 (1990).
5. Fisher, D.H.: Knowledge Acquisition Via Incremental Conceptual Clustering. Machine Learning. 2, 2, 139–172 (1987).
6. Fung, B.C.M. et al.: Hierarchical Document Clustering. Encyclopedia of Data Warehousing and Mining. pp. 970–975 Information Science Reference (2009).
7. Gennari, J.H. et al.: Models of Incremental Concept Formation. Artificial Intelligence. 40, 1-3, 11–61 (1989).
8. Gluck, M.A., Corter, J.E.: Information, uncertainty and the utility of categories. Proceedings of the 7th Annual Conference of the Cognitive Science Society. pp. 283–287 , Irvine, USA (1985).
9. Hoffman, M.D. et al.: Online Learning for Latent Dirichlet Allocation. Advances in Neural Information Processing Systems (NIPS). 856–864 (2010).
10. Jain, A.K.: Data clustering: 50 years beyond K-means. Pattern Recognition Letters. 31, 8, 651–666 (2010).
11. Katz, S.M.: Distribution of content words and phrases in text and language modelling. Natural Language Engineering. 2, 1, 15–59 (1996).
12. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons (2005).
13. Manning, C.D. et al.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK (2008).
14. Reed, J.W. et al.: TF-ICF: A New Term Weighting Scheme for Clustering Dynamic Data Streams. Proceedings of 5th International Conference on Machine Learning and Applications (ICMLA'06). pp. 258–263 IEEE (2006).
15. Rehurek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. pp. 45–50 ELRA, Valletta, Malta (2010).
16. Sahoo, N.: Incremental Hierarchical Clustering of Text Documents. Carnegie Mellon University (2006).
17. Sahoo, N. et al.: Incremental Hierarchical Clustering of Text Documents. Proceedings of the 15th ACM International Conference on Information and Knowledge Management - CIKM'06. pp. 357–366 ACM Press, Arlington, USA (2006).
18. Sahoo, N.: Three Essays on Enterprise Information System Mining for Business Intelligence. Carnegie Mellon University (2009).
19. Salton, G. et al.: A vector space model for automatic indexing. Communications of the ACM. 18, 11, 613–620 (1975).
20. Spärck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation. 28, 1, 11–21 (1972).
21. Steinbach, M. et al.: A Comparison of Document Clustering Techniques. KDD Workshop on Text Mining. IEEE (2000).
22. Tang, B. et al.: Comparing and Combining Dimension Reduction Techniques for Efficient Text Clustering. Feature Selection for Data Mining. 17 (2005).
23. Weinberger, K.Q. et al.: Feature Hashing for Large Scale Multitask Learning. 26th International Conference on Machine Learning ICML'09. (2009).
24. Zipf, G.K.: Human Behavior and the Principle of Least Effort. Addison-Wesley Press Inc., Cambridge, MA, USA (1949).