Chapter 3

Text clustering as a mining task

F. Mandreoli, R. Martoglia & P. Tiberio
Dipartimento di Ingegneria dell'Informazione,
Università di Modena e Reggio Emilia, Modena, Italy.

Abstract

In this chapter we introduce readers to the various aspects of cluster analysis performed on textual data in a mining framework. We first provide a brief overview on the techniques and the background notions on general clustering. Then, we focus on the importance and on the goals of clustering in a text mining scenario, analyzing and describing the issues which are specific to this particular field. Effective information extraction from highly dimensional textual data, clustering algorithms specifically designed to efficiently work on very large unstructured and, possibly, hyperlinked data sets, and comprehension of the clustering output are among the covered topics.

1 Introduction

Cluster analysis is a technique used to analyze a set of items and collect them into clusters, that is classes of similar items that can be treated collectively as a group. Intuitively, the items within a valid cluster are more similar to each other than they are to an item belonging to a different cluster. An example of clustering is depicted in fig. 1. The input items are represented in fig. 1(a) as points in a bi-dimensional space and the more two items are similar the more the corresponding points are close to each other; the most reasonable organization into clusters is shown in fig. 1(b) where the points belonging to the same cluster are represented with the same shape. Clustering is a widely diffused technique which has been used in a large variety of contexts from image segmentation [1] to object and character recognition [2], from information retrieval [3] to data mining [4]. Basic texts and papers on data clustering include those by Anderberg [5], Hartigan [6], Everitt [7], Kaufman [8] and Jain, Murty and Flynn [9].

WIT Transactions on State of the Art in Science and Engineering, Vol 17, © 2005 WIT Press
www.witpress.com, ISSN 1755-8336 (on-line)
doi:10.2495/978-1-85312-995-7/03

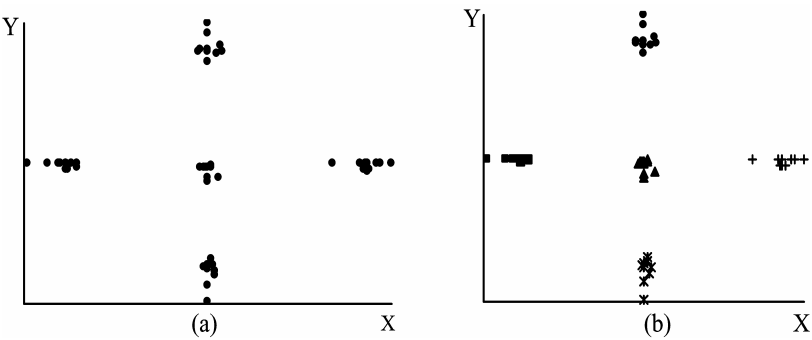


Figure 1: Data clustering.

One of the fundamental properties which characterizes cluster analysis and which constitutes the main difference with the classification task [4] is that the formed clusters are not known prior to processing, as classification implies, but are defined by the items forming them. Because of this property, cluster analysis is a useful exploratory activity which provides organization to large multivariate data sets. The rapid diffusion of technologies for computing and storage has enabled stand-alone people and organizations to collect and store huge amounts of information coming from different sources and collected for various purposes. If, in the past, we could rely on human analysts to perform the necessary analysis on interesting data sets, with large databases a simple query can easily return hundreds or thousands of matches. As the scenario has changed, it has been necessary to detect techniques for searching for useful nuggets among huge amounts of information. In such a context, cluster analysis has been recognized as a task of mining because it has the potential to reveal non-trivial relationships based on complex data [4].

Cluster analysis activity generally involves different steps. The proper grouping phase is usually preceded by the feature extraction and the proximity measure definition phases and is optionally followed by the data abstraction and the assessment of output phases [9]. Figure 2 depicts a typical sequence of three of these steps, also including a feedback step where the grouping process output could affect subsequent feature extraction and similarity computations.

The grouping of a set of items into clusters is led by some characteristics or *features* which are regarded as appropriate for representing the semantics of each item for such a specific task. For instance, images are usually represented by color histograms and/or textures, documents are often characterized by vectors of meaningful terms, and so on. Feature selection is the process of identifying the most effective subset of the original features to be used in clustering whereas feature extraction concerns the extraction of the feature values from the items. The features extracted are then used to compare the items to be clustered. In order to do it, some means of quantifying the degree of association between pairs of items is required. This is usually a *distance measure*, or a measure of similarity or dissimilarity.

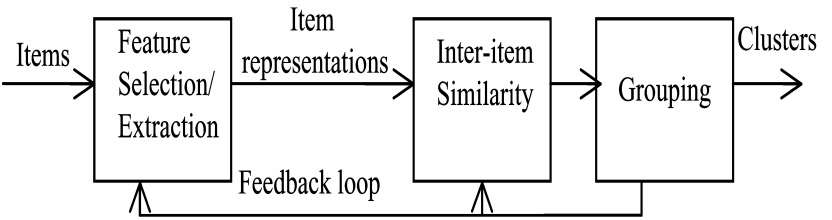


Figure 2: Stage in clustering

The variety of techniques for representing items, measuring proximity, and grouping has produced a rich assortment of clustering methods. Some of these have been specifically devised for handling data in textual form.

As the most natural form of storing information is text and the Web continues to grow at an amazing speed, the need for mastering the knowledge accumulated in huge amounts of text and hypertext documents has become more and more impelling. Thus, *text clustering* has become an appealing variation on data clustering which applies the same analysis process of data clustering to the domain of textual information. However, the specificity of the text domain and of the goals of the mining task on such kind of data has introduced new challenges. Text clustering helps to tackle the information overload problem in several ways. As a *text mining task*, it allows the organization of large text collections by summarizing at a glance their contents, thus enabling users to selectively drill deeper to explore specific topics of interest without reading every document [20]. In this way, it also allows the discovery of concepts which are latent in the analyzed documents. Thus, clustering is inherently useful, for instance, in building an ontology from a col-lection of plain texts or in disambiguating and navigating the results retrieved by a search engine or for personalized information delivery by providing a setup for routing new information such as that arriving from newfeeds [21]. In order to do it, it is necessary to face multifaceted problems specific to the text mining field.

The first challenge is that textual documents are expressions of natural language, thus easily understood by humans but not readily accessible to be used by computers. Preparing textual data in a format suitable for the cluster analysis is a central problem affecting the overall clustering process. Thus, the feature selec-tion/extraction phase of the text clustering process essentially requires complex text processing operations relying on natural language interpretation. The information needed for text clustering is hidden inside the text and such operations must be able to identify features in the text carrying such information. On the other hand, the cardinality of the feature set that can be extracted from a document collection is usually very high, easily running into several thousands. This problem affects once again the preparatory phase which must be able to select a reasonable number of meaningful features, possibly without human intervention. Moreover, when the cluster analysis enters into the heart of the clustering process, *i.e.* the grouping, it must be able to handle highly dimensional, but sparsely populated feature collections. Finally, the problem of

high dimensionality of the feature space affects the interpretability of results. In text mining, the visualization of the clustering results is fundamental as cluster descriptions should be easily assimilated by an end-user. On the other hand, most visualization techniques do not work well in high dimensional feature space.

The rest of the Chapter is articulated in two main sections. In the next section, we present the range of generic data clustering methods available and algorithms for their implementation. In the second section we cover in detail the main approaches dealing with the text clustering task. At the end of the Chapter, we include a list of selected references.

In writing this chapter, we tried to cover the most important and up-to-date solutions to the main problems related to text clustering. However, interested readers have to know that text clustering is a very dynamic field in constant evolution. Thus, we expect many other interesting solutions to be proposed in the future.

2 Overview on data clustering analysis

Text clustering stems from data clustering. In this section, we introduce the readers to the most diffused methods employed in the data clustering field. As a matter of fact, many of these approaches have been straightforwardly applied to text.

Text clustering is a variation on data clustering which applies the same analysis process of data clustering to the domain of textual information. In particular, once features have been extracted from text, the inter-item similarity and the grouping phases no longer depend on such specific data type ands solutions for generic data clustering can be applied. Those will, therefore, be the main theme of the discussion which follows.

2.1 Similarity measures

In order to use a clustering algorithm, it is necessary to be able to measure in some way the similarity between the available items. Being items represented by features, a measure of (dis)similarity between any two items from the same feature space is thus essential. In this section we briefly discuss the most common ways to calculate the *dissimilarity* between two items. In particular, we will focus on the *Euclidean distance* [5], which is the most well-known distance measure used for items whose features are continuous, then we will briefly review the *cosine similarity*, which has been proved to be particularly effective when the involved items are in textual form [10]. Both distances assume items to be represented as vectors of features in a common feature space. In the following, $\vec{x} = (x_1,..., x_m)$ and $\vec{y} = (y_1,..., y_m)$ will denote two of these vectors.

As we have already said, the most popular and widely used metric (a non-negative, symmetric function satisfying the triangular inequality) is the Euclidean distance, also known as *L_2 norm*. It is defined as:

$$L_2(\vec{x},\vec{y}) = \left(\sum_{i=1}^{m}(x_i - y_i)^2\right)^{1/2}$$

The Euclidian distance is a special case ($p = 2$) of the L_p *norm*:

$$L_p(\vec{x},\vec{y}) = \left(\sum_{i=1}^{m}(x_i - y_i)^p\right)^{1/p} \qquad (1 \le p < \infty)$$

These distances work particularly well with 'compact' or 'isolated' clusters [11]. Further, with the aid of techniques such as feature normalization, the main drawback of these metrics, that is the tendency of the largest-scaled feature to dominate the other features, can also be successfully solved [9].

The cosine measure quantifies the similarity between items by computing the cosine of the angle subtended by the two items:

$$\cos Sim(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\ \|\vec{y}\|}$$

where the dot symbol and $\|.\|$ indicate the well-known dot product between vectors and the length of the argument vector, respectively. The cosine similarity can be transformed in a distance which is a metric by applying the following formula:

$$\cos Dist(\vec{x},\vec{y}) = \sqrt{2(1-\cos Sim(\vec{x},\vec{y}))}$$

Finally, notice that most of the clustering algorithms are based on a pairwise coupling of the items to be clustered and, thus, require the distances between every pair of items. In such situations, the algorithms are typically devised not to work on the original item set but on a matrix of distance values called *distance matrix* (*DM*). Each cell of *DM* corresponds to the distance between a particular pair of items. When the distance metric is symmetric, it is sufficient to compute the lower triangular matrix, holding the $N(N\text{-}1)/2$ distance values between all possible pairs of the N available items.

2.2 Clustering techniques

In this section we provide a brief overview of the most common approaches to data clustering. Clustering techniques are usually first categorized according to the type of cluster structures they produce [9]:

- *hierarchical* clustering techniques produce a nested series of partitions, with a single cluster at the top of the hierarchy and all the clusters composed of the single starting elements at the bottom;
- *partitional* (or flat) clustering techniques produce a one level (un-nested) partitioning of the data.

Historically, partitional clustering methods have been the most popular choice in early research works, when the computational resources and the available technology were very limited [12]. Then, in the last decade, the recent

developments in computing power, along with the constant drop of data management and storing costs, made hierarchical approaches the most interesting choice, on which most of the work in data clustering has concentrated on [13].

Hierarchical clustering methods are preferable for detailed data analysis and provide more information than partitional clustering. The structure produced by a hierarchical clustering can be graphically displayed as a binary tree, called *dendogram* (see fig. 3). A dendogram graphically depicts the merging process and the intermediate clusters. The root of the tree represents the entire collection, leaf nodes represent items (observations), while each interior node represents a group (cluster). It can be broken at different levels in order to visualize different clustering levels of the data (four levels in fig. 3).

As to partitional approaches, they are less versatile than hierarchical methods, but they are preferable for their efficiency, making them the preferred choice for such scenarios characterized by very large data sets [9]. Typically, the number k of clusters has to be determined by the user ahead of time, then such methods attempt to find an approximation of the clustering, usually by partitioning the data set in some way and then reallocating items until some criterion is optimized.

Another fundamental distinction in the clustering techniques, this time relative to the structure and operation of the employed algorithm, is the following:

- *agglomerative* (or bottom-up) methods are the most common ones; they start with each of the items in individual clusters and then, at each step, group the most similar ones. The procedure ends when one cluster containing all the available items is formed or, more generally, a stopping criterion is satisfied;
- *divisive* methods begin with all items in a single cluster and then perform splitting until the desired number of clusters is reached or another stopping criterion is met.

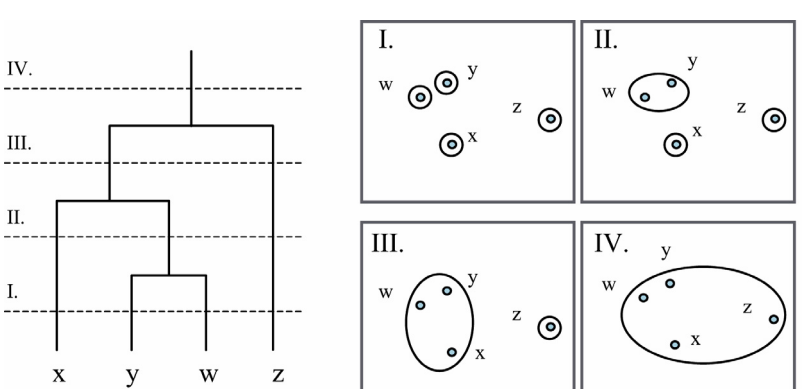


Figure 3: Example of the dendogram of a hierarchical classification.

Theoretically, such distinction is orthogonal w.r.t. the hierarchical/partitional one and, thus, leads to the distinction among four classes of clustering methods: hierarchical agglomerative, hierarchical divisive, partitional agglomerative and partitional divisive. In the discussion, we will consider the agglomera-stive/divisive distinction only for hierarchical methods, where it is more significant, whereas we will no longer make use of such a distinction for the partitional methods.

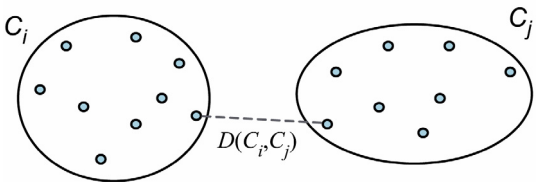
In the following, we present an overview of the most common and effective algorithms proposed for data clustering: *single-link* and *complete-link* for hierarchical clustering, and *k-means* for partitional clustering.

2.2.1 Single-link and complete-link hierarchical methods

Single-link [14] and complete-link [15] are two of the most used hierarchical clustering methods, the ones from which most of the hierarchical clustering approaches derive. Their most popular versions are implemented with hierarchical agglomerative clustering (HAC) algorithms, *i.e.* starting with all items in a separate cluster, both generate a hierarchy of partitions bottom up.

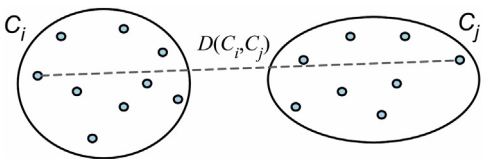
In order to perform their task, the algorithms assume that, given the distance measure $d(\vec{x},\vec{y})$ between single items, a *cluster distance function* $D(C_i,C_j)$ is defined for determining the similarity between whole groups of items.

- In single-link clustering, the distance between two clusters is the distance of the two closest items in the clusters. Thus, intuitively, such method searches over all pairs of items that belong to two different clusters the pair with the smallest distance.



$$D(C_i,C_j)=\min_{\vec{x}\in C_i,\vec{y}\in C_j} d(\vec{x},\vec{y})$$

- In complete-link clustering, the distance between two clusters is the distance of the two most distant items in the clusters. Thus, intuitively, the method searches over all pairs of items that belong to two different clusters the pair with the highest distance.



$$D(C_i,C_j)=\max_{\vec{x}\in C_i,\vec{y}\in C_j} d(\vec{x},\vec{y})$$

Regardless of the particular cluster distance function employed, the two approaches share the same basic working procedure. Given a set of items to be clustered, the procedure can be divided in steps and summarized as follows [18]:

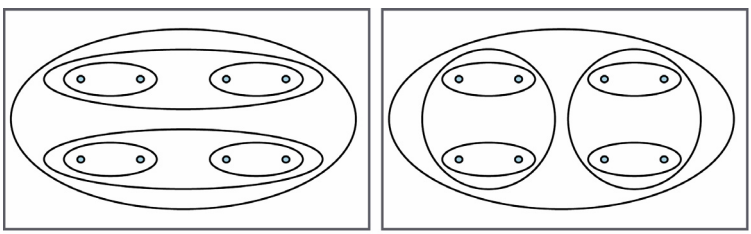
1. place each item in its own cluster;
2. compute the distance between all pairs of clusters, *i.e.* calculate the distance matrix *DM*, where each entry *DM[i][j]* corresponds to the distance $D(C_i,C_j)$ between the two clusters C_i and C_j;
3. merge the closest two clusters (*i.e.* the least distant ones);
4. update the distance matrix *DM* with the distances between the new and the remaining clusters;
5. repeat steps 3 and 4 until the stopping criterion is met (typically, one all inclusive cluster is formed).

Single-link clustering is generally more versatile than complete link in detecting particular item configurations, such as concentric clusters. However, it often results in long and thin clusters which, depending on the particular domain of application, could be inappropriate (*e.g.* see fig. 4 (a)). On the other hand, clusters generated by complete-link are typically more tight and spherical [16] and, ultimately, preferable for most text clustering applications (see fig. 4 (b)).

Depending on the particular algorithm chosen for the implementation and on the particular way the distance matrix is stored and updated, the computational requirements of single and complete-link hierarchical agglomerative clustering range from $O(N\ logN)$ to $O(N^5)$, where N is the number of items to be clustered [13]. Finally, notice that single-link has an efficiency advantage w.r.t. complete-link: It does not need to re-calculate the similarity matrix during processing, because it defines the distance between the two clusters as the distance between the closest pair of items each of which is in one of the two clusters [13].

2.2.2 K-means partitional methods

Among the huge number of partitional techniques proposed in the literature, K-means is the most widely known and used in several domains. As with all partitional methods, also this method requires as input the desired number of output clusters *k*, giving rise to the problem of choosing it in the most effective way [17]. Having specified *k*, the method randomly chooses *k seeds*, then it forms *k* initial



(a) Single link clustering results. (b) Complete link clustering results.

Figure 4: Hierarchical clustering method examples.

clusters and, iteratively, reallocates the items to different clusters in order to improve the overall result. The stopping criterion can be reaching a desired number of iterations or, more usually, meeting a convergence criterion, *i.e.* reaching the point where there is no reassignment of any item from one cluster to another [9].

While hierarchical methods depend on a distance function between clusters, typical partitional methods, and in particular K-means, depend on the notion of a cluster *centroid*. The centroid $\mu(C)$ of a cluster C is the mean of the group of points forming the cluster:

$$\vec{\mu}(C) = \frac{1}{|C|}\sum_{\vec{x}\in C}\vec{x}$$

The assumption behind a centroid is that such point can significantly represent an entire cluster, and that the distance between centroids can approximate well the distance between the clusters they represent. Note that a centroid does not necessarily correspond to any actual item, in fact it almost never does.

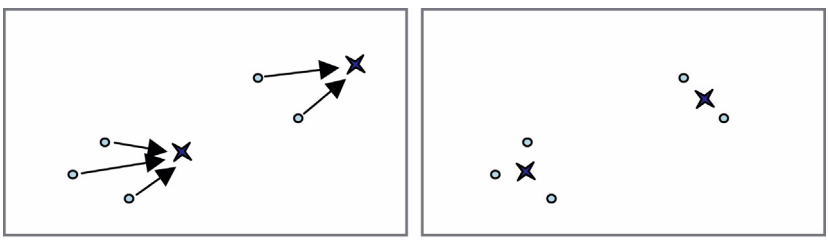
The K-means method follows these steps [18]:

1. select k initial centroids (seeds);
2. assign all items to the closest centroid (*assignment* phase);
3. update the centroid of each cluster (*recomputation* phase);
4. repeat steps 2 and 3 until the stopping criterion is met (typically, until convergence is reached).

Figure 5 depicts a simple example of the key phases in K-means clustering, for $k = 2$. In the assignment phase (a) the items are assigned to the closest initial centroid (or seed, represented with a star), then the correct centroid is recomputed for each of the two clusters (b).

Notice that the initial choice of the seeds is particularly important in this method and directly influences the result. In other words, poorly chosen seeds generally result in a sub-optimal clustering.

The K-means algorithms available in literature present a much lower time and space complexities than the hierarchical ones. In particular, their time complexities are typically linear in all relevant factors, that is the number N of items to be clustered, the desired number of clusters k and the number of iterations required [19].



(a) assignment phase (b) recomputation phase

Figure 5: K-means clustering method example.

3 Problems and solutions in the text clustering field

Text clustering is used to discover 'latent concepts' in sets of unstructured text documents and to summarize and label such collections. Clustering large sets of text documents in a data mining framework places the following special issues on clustering techniques, motivating the need for developing new solutions:

3.1 Effective extraction of meaningful features from plain texts

Often the text documents themselves cannot be directly used for text clustering, but are represented in the system by document surrogates. On the other hand, plain text is written for human readers and it is not readily accessible to be used by computers. This requires solutions for the extraction of meaningful features based on the natural language interpretation of plain text.

3.2 Effective treatment of high dimensionality

The cardinality of the feature set that can be extracted from a document collection is very high, easily running into several thousands. On the other hand, it is not meaningful to look for clusters in such a high dimensional space as the average density of points anywhere in the data space is likely to be quite low [22]. This requires solutions for dealing with sparsely populated data vectors or for projecting vectors into small subspaces.

3.3 Interpretability of results

Text mining applications usually require cluster descriptions to guide end-users in the process of browsing the clustering. Thus it is particularly important to have simple representations because most visualization techniques do not work well in high dimensional spaces [22].

3.4 Efficiency and scalability of the clustering process

From the text mining perspective, clustering must be able to deal with huge amounts of data. To effectively group large collections of text documents, the clustering algorithms must be efficient and scalable. That is, the running time of a text clustering algorithm must be predictable and acceptable in large data sets.

Most text clustering algorithms rely on the so-called *vector-space model*, which has already been introduced in Chapter 1. The basic idea is to extract unique content-bearing terms from the set of documents treating these terms. Thus, the number of terms used in the collection defines the dimension of the term space where each document d is represented as a vector osf weights, one for each term: $d=(w_1,\ldots, w_m)$.ss These term-weights must express the 'importance' of the corresponding terms in the document and are ultimately used to compute the degree of similarity between pairs of documents. There are a number of possible measures for computing the similarity between documents, but the most diffused one is the cosine similarity,

which has been introduced in subsec. 2.1. By applying the cosine similarity to two vectors of term-weights $d_i=(w_{1,i},\ldots,w_{m,i})$ and $d_j=(w_{1,j},\ldots,w_{m,j})$, the resulting formula is

$$\frac{\sum_{h=1}^{m} w_{h,i} w_{h,j}}{\sqrt{\sum_{h=1}^{m} w_{h,i}^2} \sqrt{\sum_{h=1}^{m} w_{h,j}^2}}$$

With reference to the stages in clustering shown in fig. 2, the emphasis in this section will be on the range of solutions available for the above mentioned problems, thus resulting in interesting techniques for text clustering analysis. Moreover, a separate section is devoted to the problem of clustering hypertext documents. As the Web is nowadays a sheer mine of HTML/XML documents and the number of hypertext documents managed by organizational intranets is rapidly growing, text clustering helps information retrieval systems to organize this vast amount of data and to make the retrieved search results easier to browse. On the other hand, the hypertext nature of such documents has posed new issues in the text clustering fields and new solutions have been consequently proposed.

3.5 Feature selection and reduction

The primary step to cluster a document set is to prepare documents in a format suitable for clustering analysis. In the feature selection task, it is important to identify features in the text that carry useful information from documents. As the most diffused model for text clustering is the vector-space model, the feature selection task maps each document into a vector of term-weights. The problem of extracting features from documents has been extensively studied by the IR community [3] and by a large body of linguistics and computational linguistics. A good discussion on this topic is also given in Chapter 1. In this Chapter, we review the main aspects of this process. Each document usually undergoes a pre-processing phase where it is tokenized using simple syntactic rules and, after having deleted such tokens belonging to a stopword list, the remaining tokens are stemmed to canonical form. Each canonical token represents an axis in the vector space model. The weights of terms can be assigned in various ways. The most common is the *tf x idf* weighting scheme, where *tf* stands for term frequency and represents the term's frequency within the document, whereas *idf* stands for inverse document frequency and expresses the importance of the term in the document w.r.t. the entire collection. The motivation for usage of an *idf* factor is that terms which appear in many documents are not very useful for charactering a specific document. The most diffused way to use the *idf* factor is $log_2(N/n)$ where n is the document frequency, *i.e.* the number of documents in which the specific term appears whereas N is the total number of documents.

The main drawback related to such a general approach is the prohibitively high dimensionality of the feature space since even after preprocessing there are typically still several thousands of terms. Moreover, due to the high dimensionality, most weights are zero for a single document. In other words, vectors are often both very big and sparse. Usually, clustering algorithms do not scale well to a high-dimensional feature space. In other words, the bottleneck in clustering text documents is calculating the distance between term vectors. One possible solution to speed up clustering is to project each document onto a small subspace of the term space, thereby reducing the average number of terms in

each document [28]. To this end, feature reduction methods [25] take each vector of term weights $d=(w_1,...,w_m)$ in the native m feature space and convert some non-zero values to 0, possibly first modifying the vector d in an arbitrary way. The *document projection* can be performed in two ways: locally or globally [32]. For ease of reference, before discussing the two approaches we refer readers to table 1 which summarizes the used symbols and their meanings.

The local projection, also known as *truncation*, removes from each document its less 'important' terms by setting to 0 a fixed number of weights, usually the smallest ones. It is called local as it projects each vector onto a different subspace. For this reason, mainly dense and large vectors, such as centroids, benefit from truncation while document vectors, which are usually quite sparse, benefit minimally from the sparsification provided by truncation.

The alternative to local projection is global projection, also known as *dimension reduction*, which consists in reducing the native space by first choosing the terms to be deleted and then deleting these terms from each document vector.

One of the most diffused approaches for reducing the space dimensionality is feature ranking because of its simplicity, scalability, and good empirical success. It removes non-informative terms and usually relies on term collection statistics measuring, under different aspects, the significance of each term in the collection. In the range of the proposed techniques, document frequency filtration is the simplest one. It consists in computing the number of documents in which a term occurs and in removing from the feature space those terms whose value does not satisfy specific requirements [23, 24]. In [23], for instance, they remove terms whose document frequency is less than some predetermined threshold. Here the basic assumption is that rare terms are either non-informative or not influential in global performance. In more general frameworks, we can introduce a significance function s, so that the significance of each term t is given by $s(t)$. In this way, a term ranking is possible and the first l best quality terms define the dimension l of the resulting vector space. Various significance functions can be defined. Following the ideas of Salton and McGill [26], the significance of the term t can be measured by (tf_h is the frequency of term t in the document d_h)

$$s(t) = \sum_{h=1}^{N} tf_h^2 - \frac{1}{N}\left(\sum_{h=1}^{N} tf_h\right)^2$$

which is proportional to the term frequency variance. An interesting variation of the above formula is given in [24] where the total number N of documents is substituted by the number of document N_t in which t occurs at least once. Such a variation penalizes those terms that occur in many documents only once and thus that

Table 1: Symbols.

Symbol	Meaning
N	Number of documents
m	Dimension of the original feature space
tf	Term frequency
idf	Inverse document frequency
n	Document frequency

may lack any specificity. More sophisticated approaches estimate the term significance based on how commonly a term is likely to appear in 'closely-related' documents [27] or on the fact that terms that occur in the same context are equivalent [24].

Another category of dimension reduction approaches exploits the vector space model by means of linear transformations to documents and terms, regarded as vectors in an Euclidean space [31]. Among those, the Latent Semantic Indexing (LSI) and the random projection are the subject of the discussion which follows.

LSI maps the documents from the term space to an orthonormal semantic latent subspace by grouping similar documents together, as well as similar terms together. Each group corresponds to a principal component or concept as it has a high discriminatory power and is orthogonal to the other such groups. LSI applies linear algebra transformations to the term-by-document matrix W obtained by putting in each row the term vector of each document. In this way, each element w_{ij} is the weight of the term t_i in the document d_j and, if two terms are related, we would expect them to occur in similar sets of documents, hence the rows corresponding to these terms should have some similarity. The main objective of LSI is thus to discover such rows and/or columns which are somewhat 'redundant' and to delete them. In order to do it, by means of the Singular Value Decomposition (SVD) it factorizes W into three matrices TCD^T, where C is the concept matrix, T is the matrix of term-concept similarity and D is the matrix of concept-document similarity. Algebraically speaking, $C = diag(\lambda_1,...,\lambda_r)$ is a diagonal matrix where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_r$, and T and D are column-orthonormal matrices. Notice that being C a diagonal matrix, each weight w_{ij} depends on the latent concepts:

$$w_{ij} = \sum_{k=1}^{r} t_{ik} \lambda_k d_{jk}$$

LSI deletes the smaller lambda values by retaining only the first predetermined k bigger singular values $\lambda_1,...,\lambda_k$ with the corresponding rows of T and D, which induces an approximation of W, denoted as $W_k = T_k C_k D_k^T$. It has been shown that W_k is the closest k-dimensional approximation to the original term-document space represented by the matrix W. Each row of T_k represents a term as a k-dimensional vector, similarly each row of D_k represents a document as a k-dimensional vector. In text clustering, LSI can thus be used to preprocess the data for clustering and visualization of document collections in very low-dimensional spaces (2- or 3-d). LSI is able to reduce with a good approximation the space to 100–300 terms in front of spaces of thousands of terms. More details on LSI can be found in [28].

Random projection relies on a theorem which establishes that a projection of a set of points to a randomly oriented subspace of suitably high dimension induces small distortions in most inter-point distances with high probability. More precisely, Deerwester *et al* [29] show that it is verified if the dimension k

of the subspace is chosen between $\Omega(\log N)$ and $O(\sqrt{N})$. Although such a random projection can be used to reduce the dimension of the document space, it does not bring together semantically related documents. LSI, on the other, hand seems to achieve the latter, but its computation time is a bottleneck ($O(Nmm)$ or $O(NNm)$ in the worst case). For this reason, in Papadimitriou *et al* [30] a two-step approach is proposed, whose final representation is very close to that of the direct application of LSI. It consists in applying a random projection to the initial corpus to l dimensions, for some small $l > k$, to obtain, a much smaller representation, which is still very close (in terms of distances and angles) to the original corpus. Then, LSI is applied to the resulting l-space.

Finally, as already mentioned before, projecting documents usually introduces an approximation in the document representation. In [32] two projection approaches, truncation and LSI, have been evaluated by using the Euclidean distance and a variation of the k-means algorithm. They showed that such approaches offer a dramatic advantage over full-profile clustering in terms of time efficiency and that such improved efficiency, surprisingly, is accompanied by no significant difference in the cluster quality. Thus, in contrast to similarity search, clustering can proceed successfully even if document representations have been reduced at a considerable loss of information, as clustering is a less fine-grained task than similarity search and therefore it requires less precision in determining the distances of items with respect to each other.

3.6 Efficient clustering of large unstructured data sets

In recent years, applications requiring to cluster very large collections of items have become more and more popular. In particular, in document information retrieval, modern text mining applications work on millions of documents, each described by sparse vectors having a dimensionality of hundreds of features. In such a scenario, it is evident that only a few of the standard clustering approaches can be successfully employed. For example, as seen in section 2, hierarchical algorithms are the most versatile approaches to clustering but, due to their high time and space complexities, they are clearly not sufficiently scalable w.r.t. large and high-dimensional data sets. On the other hand, K-means is much more popular in this domain and, in several variants, has been the right choice to successfully cluster very large data [9], also including textual data [18, 44, 45]. In general, much research work has been done to develop variants of the existing algorithms and completely new techniques, which are able to work effectively and efficiently on such large data sets. In the next subsections we will describe the main contributions on this topic.

3.6.1 K-Means clustering variants

The most exploited K-means variant for textual data is the *spherical K-means* [21], which employs the standard K-means algorithm described in section 2 together with the cosine similarity between the document feature vectors. Spherical K-means has several advantages:

- linear time complexity in the size of the data set. In particular, it converges quickly to a local maximum [21];
- linear space complexity, thus requiring a reasonable amount of space to store the data matrix [9];
- exploitation of the sparsity of the feature vectors, since the cosine similarity is particularly simple and fast to compute for sparse vectors [21].

On the other hand, the spherical K-means approach also shares some of the drawbacks of the original K-means technique. Most notably, its results' quality is not always as high as expected, and is particularly sensitive to the initial seed selection. Futher, the number of desired clusters k has to be pre-determined by the user.

The above considerations have given rise to the need of further variants of the K-means approach. In the last decade, several alternative approaches have been presented, delivering higher quality results with large data sets while keeping the time complexity linear. We will now briefly analyze such variants. The improvements concern different aspects or phases of the K-means algorithm:

- initial seed selection;
- recomputation of the clusters' centroids.

Initial seed selection in the standard K-means technique usually involves random selection of items. Such an approach produces different results at each application and, generally, does not guarantee particularly good quality in the resulting final clusters. In order to improve the initial seed selection, two ways exist. One way is to choose a small random sample of the documents of size $\sqrt{kN}$. Then, a hierarchical clustering subroutine is applied to such items and the centroids of the clusters found are chosen as the initial seeds. Such technique is known as *buckshot* [44] and achieves linear time complexity. Another way to find the k initial seeds is to break the item collection into N/m buckets of a fixed size $m > k$ and to apply the clustering to each of those buckets separately. Then, the clustering subroutine is applied again on the centroids of the newly created groups. The iteration terminates when only k groups remain. The centroids of such groups will be chosen as the initial seeds. Such method is known as *fractionation* [44] and, again, maintains overall linear time complexity while improving the successive clustering result.

In K-means, seed selection is followed by some number of centroid adjustment iterations. In the standard algorithm, centroids are adjusted only at the end of an iteration, *i.e.* when all items have been assigned to the closest centroids. In order to improve this step of the algorithm, it is possible to update centroids after each item is assigned. This simple variant is known as *continuous K-means* [45, 47], while the standard version is also called non-continuous. Continuous updates are proven to be more effective, *i.e.* produce higher quality clustering results.

The K-means variants presented so far involve enhancements in the various steps of the original clustering method. Another good means of improving the K-means effectiveness without affecting its efficiency is to introduce an extra

'refinement step' at the end of each main cycle of the algorithm, *i.e.* after the assignment and recomputation steps. Before proceeding with the next centroid assignment and update iteration, such *cluster refinement* step improves the quality of the current partitioning by means of particular auxiliary techniques. In order to understand how such an improvement is provided and why it is particularly needed, we need to understand the concept of *partition quality function* $F(P)$, which quantifies the effectiveness of a given partition $P=(C_1,...,C_k)$ (composed of k clusters) of the given item set. The goal of each clustering algorithm can be interpreted as the maximization of such function, which, thus, will also be called *objective function*. In particular, with K-means this function expresses the concept of *average pairwise similarity*, *i.e.* it is particularly high for partitions involving 'coherent' clusters, whose elements are very similar to one another. We call such function $F_{APS}(P)$, where *APS* stands for average pairwise similarity. Its computation is particularly fast when the chosen similarity metric between items is the cosine similarity and the feature vectors are normalized, *i.e.* they have length 1, as in a typical document clustering scenario:

$$F_{APS}(P) = \sum_{C\in P}\left(\frac{1}{|C|^2}\sum_{\vec{x},\vec{y}\in C}\cos Sim(\vec{x},\vec{y})\right) = \sum_{C\in P}\left(\frac{1}{|C|}\sum_{\vec{x}\in C}\vec{x}\cdot\frac{1}{|C|}\sum_{\vec{x}\in C}\vec{x}\right) =$$
$$= \sum_{C\in P}\left(\vec{\mu}(C)\cdot\vec{\mu}(C)\right) = \sum_{C\in P}\left\|\vec{\mu}(C)\right\|^2.$$

Then, the average pairwise similarity function can be computed straightforwardly, by simply summing the squares of the lengths of the centroid of each cluster in the partition. Each iteration of the K-means algorithm aims at increasing such objective function value, eventually reaching the ideal situation, *i.e.* the maximum of the function. However, depending on the particular situation, the standard algorithm could 'get stuck' at a local maximum and thus would be unable to find the best partition for the given problem. At this point, the need for refinement becomes clear. One possible refinement technique is known as *first variation* [46] and is aimed at improving the partition at the end of each K-means iteration. A first variation step moves a single item from one cluster to another. The moved item is the one producing the highest increase in the objective function value of the overall partition. Multiple iterations allow the overall algorithm to escape from local maxima, so that new iterations of K-means can be applied to further increase the objective function and, thus, the final quality of the produced clusters. Another refinement method is to break in two each of the clusters available at the end of a K-means iteration. The break can be performed by applying K-means clustering with $K = 2$ to the items in each of the clusters. Then, the closest pairs, thus the ones producing the best enhancement in the objective function, are rejoined. Such improvement is known as *split and join* [44] and, again, can be applied more than once per cycle (more split and join processes), in order to improve the quality of the partition.

A final variation on the K-means 'theme' is a technique known as *bisecting K-means*, a simple variant that has been proved to produce results of particular high quality in large data sets scenarios, again maintaining linear time complexity [18]. In the document clustering domain, the clusters obtained with such variant are generally as good as those produced by the much more computationally expensive agglomerative hierarchical techniques. The algorithm follows these steps:

1. start with all items in one single cluster;
2. choose a cluster to split;
3. search for a possible split in 2 sub-clusters of the chosen cluster by using basic K-means algorithm with $K = 2$ (bisecting phase);
4. repeat the search of step 3 for a given number of times, then actually perform the split that produces the highest quality partition;
5. repeat steps 2, 3, and 4 until the desired number of clusters is reached.

As to step 2, the cluster to split can be chosen with various criterions: It can be the least 'coherent' one (the one with lowest intra-cluster similarity), or more simply the largest one. As to step 4, we remind that the 'quality' of a partition can be quantified by means of the partition quality function: Higher values of such a function correspond to higher quality in the partitioning.

3.6.2 Relational data analysis (RDA) clustering

There are a number of clustering techniques that go beyond K-means in terms of flexibility, however, they often do not have the linear complexity required in high dimensionality contexts. *RDA* (*Relational Data Analysis*) clustering, also known as *binary relational* clustering, is one of the few techniques that is not a K-means variant, not requiring a priori the number of desired clusters K, but shares with K-means its optimal scalability and efficiency, making it one of the most exploited techniques in the text clustering area. As with all other clustering techniques, also RDA tries to find a partition that optimizes the chosen partition quality function $F(P)$. As we have seen, the K-means $F_{APS}(P)$ function measures the average pairwise similarity inside the clusters and tries to maximize such value. Having such an objective, if not coupled with the constraint of generating k clusters, it would favor larger number of clusters, reaching optimality with one cluster per available item. The function employed by RDA, $F_{NCC}(P)$, is expressed in terms of the so-called *New Condorcet Criterion* [48], which tries to maximize not only the coherence of the clusters, but also the dissimilarity between them. As a consequence, the optimal number of clusters is the output, not the input, of the clustering algorithm. RDA and NCC work with categorical (binary) vectors of features. Such vectors basically express if each of the available features is present (value 1) or is not present (value 0) in a given item. For each categorical feature, the distance is 0 if the involved features have equal values (they share a given feature) and 1 otherwise. The distance $d(\vec{x}, \vec{y})$ between two items $\vec{x}$ and $\vec{y}$ is thus the number of features for which the two items have different values. Then, the partition quality function can be expressed as:

$$F'_{NCC}(P) = \sum_{C \in P} \sum_{\vec{x} \in C} \left(\sum_{\vec{y} \in C} \left(m - d(\vec{x}, \vec{y}) \right) + \sum_{y \notin C} d(\vec{x}, \vec{y}) \right)$$

where m is the dimension of the items' feature vectors. The two sums in the brackets express, respectively, the number of intraclass 'agreements' (number of features with the same value between items in the same cluster) and the number of interclass 'disagreements' (number of differing features between items in different clusters). By viewing the distances $d(\vec{x}, \vec{y})$ as the elements $d_{x,y}$ of a distance matrix DM and by representing the partition by an equivalence relation using a binary matrix $E = [e_{x,y}]$, where $e_{x,y}=1$ if x and y belong to the same cluster, the function can be easily rewritten as:

$$\begin{aligned} F'_{NCC}(P) &= \sum_{\vec{x} \in P} \sum_{\vec{y} \neq \vec{x}} \left(d_{x,y}(1 - e_{x,y}) + (m - d_{x,y}) e_{x,y} \right) = \\ &= \sum_{\vec{x} \in P} \sum_{\vec{y} \neq \vec{x}} (m - 2d_{x,y}) e_{x,y} + \sum_{\vec{x} \in P} \sum_{\vec{y} \neq \vec{x}} d_{x,y} \end{aligned}$$

The second term does not depend on the given partition, but is a constant with respect to the given elements. Then, the partition quality function can be expressed as:

$$F_{NCC}(P) = \sum_{\vec{x} \in P} \sum_{y \neq x} A_{x,y} e_{x,y}$$

where $A_{x,y} = (m\text{-}2d_{x,y})$. With this objective function to be maximized, we see that RDA clustering is none other than an integer programming optimization problem [48], where the Boolean variables $e_{x,y}$ are subject to standard integer, symmetry and transitivity constraints. The problem can be solved by means of integer programming [48], requiring exponential computation time, linear programming [49], reducing the optimality of the results, or by means of heuristic algorithms. A heuristhic algorithm by Michaud [50] has proved to be particularly effective and efficient, running in linear time and producing very close to optimal solutions; such a method has been incorporated in IBM Intelligent Miner (Demographic Clustering algorithm).

3.6.3 New clustering approaches from the DataBase community

Recent research in the DataBase and DataMining communities has made available a large number of further approaches to the problem of clustering very large data sets. Such approaches do not only focus on scalability and, in particular, on the minimization of the computational complexity, but also on the optimization and minimization of CPU time and I/O costs, which can be particularly critical with data sets of millions of items.

In order to solve the clustering problem, the approaches we present in this subsection find new and efficient ways to 'explore' the high dimensional space of original data items in order to 'discover' an optimal partition for them.

Techniques such as *random sampling* (in which items are randomly selected from a convenient space) and *heuristic* search algorithms (algorithms that produce a solution which is not exact but that approximates well the optimal one) are exploited, often based on ad-hoc spatial access methods and data structures such as graphs or tree indexes [51, 52].

The problem to be solved is, as always, to find the optimal partition of the given items. Among the proposed approaches, one of the most simple but effective ideas is to restrict the search to partitions involving a specified number k of clusters and to find the best partition by means of navigating an appropriately constructed graph [51]. In particular, a graph $G_{N,k}$, N being the number of items and k the desired number of clusters, is built, where each node represents a solution (partition) for the clustering problem and is indicated as a set of k objects. Such objects are the centroids of each of the k clusters in the final partition. Two nodes are connected by an arc (*i.e.* they are neighbors) if their sets differ by only one item. Each node of the graph is assigned a cost, that can be defined as the total distance between every object in each cluster and the centroid of the respective cluster. Then, the clustering algorithm can be viewed as the search for a minimum node on the graph $G_{N,k}$. One effective way of exploring such graph is the one employed in the CLARANS algorithm [51]: starting from a randomly chosen graph node, it does not explore all its neighbors to find the minimal cost one, but only checks a random sample of the available neighbors. Because of the use of random sampling, the time complexity is $O(N)$, linear in the size of the items set.

A more sophisticated way of organizing and exploring the solution space, *i.e.* the space where each point represents a solution to the clustering problem, is to exploit dynamic indexing structures, that allow to reduce not only the computational complexity but also the I/O costs required by the clustering algorithms. One of the most significant of such approaches is BIRCH [52], an algorithm that employs a balanced tree structure called *CF Tree*, where summary information about candidate solutions' clusters are stored. The leaf nodes of such tree are organized hierarchically up to the root. The key data in a CF tree is the *clustering feature* (*CF*). A clustering feature vector $CF = (NC, \overrightarrow{LS}, SS)$ is a triple summarizing the information that is maintained for each cluster, that is the number NC of items in the cluster, the linear sum $\overrightarrow{LS}$ and the square sum SS of the NC items. Each leaf node contains one or more CF entries. Each internal node of the tree contains a certain number of pointers to child nodes, together with the respective CFs. In general, each node represents a cluster made up of all the subclusters represented by its entries (leaf node) or by its children entries (internal node). The BIRCH clustering is composed of two main phases [52]:

1. load data items into memory by dynamically building the CF tree;
2. apply an agglomerative hierarchical clustering directly to the subclusters represented by their CF vectors.

After phase 1, the actual clustering phase (phase 2) is performed very efficiently, since it does not require I/O operations (the CF tree is maintained

in the main memory). Further, the problem of clustering original data is reduced to a smaller problem, *i.e.* clustering the subclusters in the leaf entries of the CF tree.

The methods described up to this point are very effective in identifying clusters that exist in the original items' space. Since these methods work on the whole space and since such space has usually a very high dimensionality, before applying them it is often desirable to perform one of these two preliminary operations: Employ one of the dimensionality reduction methods (see previous section) or manually specify a subset of the original space (a subspace) to be analyzed for clustering. The latter option is clearly not sufficiently flexible and is quite error-prone, while the former is a much better choice but may present some shortcomings. First the new 'reduced' dimensions could be difficult to interpret and it would be hard for the user to understand the produced clusters in relation to the original data space [22]. Further, the reduction would be 'static' and would not allow the clustering algorithm to find all the clusters that may exist in the different subspaces of the original space. If this is what is required, then the clustering technique must be able to automatically find subspaces with high density clusters, *i.e.* subspaces with a high number of close data items. Such a technique has been recently proposed and is called CLIQUE [22]. It consists of the following main steps:

1. identification of subspaces that contain clusters;
2. identification of the clusters.

In order to perform step 1, the algorithm first partitions the original item space in non-overlapping rectangular *units*, obtained by partitioning every dimension into a user-specified number ξ of intervals of equal length. Then, a bottom up algorithm identifies *dense* units, *i.e.* units for which the fraction of the contained items is above a specified density threshold τ. Figure 6 depicts a simple two-dimensional example, where the two dimensions are each divided in six equal size intervals. The dense units are shown as shaded squares. The algorithm proceeds level by level, first making one pass over the items' data and identifying 1-dimensional dense units. Then, the generic i-th step receives as input D_{i-1}, the set of all $(i-1)$-dimensional dense units, finds in a pass the set C_i all the candidate i-dimensional dense units, then discards those which have a projection in $(i-1)$ dimensions that is not included in D_{i-1}. The procedure is based on the property that for any i-dimensional dense unit, its projections in any of the $i-1$ dimensions must also be dense [22].

The input to the second step is the set of the dense units D, all in the same i-dimensional space. The desired output is the partition of D into k clusters $C_1,...C_k$. Each cluster will be composed by a set of connected units, which in turn contain the original items. Two units u_1, u_2 are *connected* if they have a common face or if there exists another unit u_3 such that u_1 is connected to u_3 and u_3 is connected to u_2. Each cluster must be determined in such a way that all its units are connected and no two units from different clusters are connected.

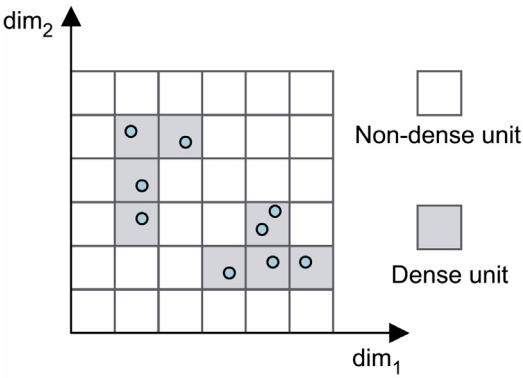


Figure 6: Identification of dense units in a two-dimensional subspace.

For instance, in the example shown in fig. 6, the number of identified clusters would be two. In general, in order to determine the clusters, this simple search algorithm can be employed:

1. take a unit *u* in *D* and assign it to a new cluster;
2. find all the units *u* is connected to and assign it to the current cluster;
3. if there are still units in *D* which have not yet been visited, repeat the procedure from step 1.

As with the other clustering algorithms for highly dimensional data, also the CLIQUE algorithm scales linearly with the number of input items. Further, it has a good scalability as the number of dimensions (features) is increased [22].

3.6.4 Document-specific clustering approaches

Up to this point, we have presented the most used and effective algorithms used to cluster very large and highly dimensional data sets. In particular, such algorithms are the most used also in the context of text and document clustering, however, they are completely general and can, therefore, be applied to many other contexts. Besides such general purpose approaches, many techniques have been proposed which are devised *ad hoc* for text clustering, thus their basic dynamics and output are specifically tuned and benefit from the unique peculiarities of this particular and complex scenario. Of particular interest is a technique named *Frequent Term-based Clustering (FTC)* [57], which is not based on the vector space model but uses the concept of *frequent term sets* for text clustering. Frequent term sets are sets of terms co-occurring in many documents of the given document set. A well-selected group of frequent term sets can be considered as a clustering. In fact, each frequent term set represents *per se* a cluster, or better it actually constitutes its textual description, being the set of the words which appear more frequently in the involved documents. Note that such description greatly facilitates the user in understanding the clustering results, fulfilling one of the most important requisites of a good clustering

algorithm. The desired clustering is the one which covers the whole document set, *i.e.* each document appears in at least one cluster, and whose clusters are the least overlapping in their contents, *i.e.* they do not share many common documents. More formally, indicated by $DocSet=\{Doc_1,\ldots,Doc_N\}$ the document collection and by $F=\{F_1,\ldots,F_k\}$ a set of frequent term sets in *DocSet*, the *overlap* $Ov(C_i)$ of a cluster C_i with respect to the other clusters can be defined as [57]:

$$Ov(C_i) = \frac{\sum_{Doc_j \in C_i} (f_j - 1)}{|C_i|}$$

where $f_j = \left|\left\{F_i \mid F_i \subseteq Terms\left(Doc_j\right)\right\}\right|$ is the number of all frequent term sets 'supported', *i.e.* whose terms are all present, in document Doc_j.

The FTC algorithm works in a bottom-up fashion and follows these steps:

1. start with an empty result set *RIS* and the set $FT = \{t_1, \ldots, t_m\}$ of frequent terms in *DocSet*;
2. compute the set F of all possible sets of frequent terms extracted from FT (*i.e.* $F = \wp(FT)$). For each frequent term set F_i in F compute the corresponding cluster C_i as the set of documents in which the terms in F_i appear and the overlap $Ov(C_i)$;
3. select the element F_i from F whose cluster has the lowest overlap with the other candidate clusters and put it in *RIS*;
4. remove the chosen element F_i from F and the documents covered by it from *DocSet*;
5. repeat steps 2–4 until the frequent term sets of *RIS* cover the entire *DocSet*.

The FTC algorithm can also be applied hierarchically (*Hierarchical FTC*, or *HTFC*) to a document collection. Standard FTC is first applied only to 1-level term sets, *i.e.* the term sets composed of single terms, then the resulting clusters are further partitioned on the frequent term sets of the next level, and so on. The obtained hierarchical tree can then be easily browsed by the user (see section 3.2.6 on clustering output for an example).

3.6.5 A short note about memory management and distribution

It is important to understand that many of the algorithms that we have presented are applicable to large data sets 'as they are' only providing that it is possible to accommodate all the input items' data, *i.e.* all the feature vectors, in the main memory. If the size of the data set is particularly large and it is not possible to employ a particular algorithm because of its too high space requirements, some techniques can be employed in order to solve this problem. In particular, these are the main possible approaches [9]:

- *Initial random sampling* approach: A subset of the item set is randomly chosen, and the clustering algorithm is applied only to this subset instead of to the whole database.
- Advanced *data summarization* methods: When applied to data items, they produce high-quality 'micro-clusters' to efficiently and effectively support the clustering algorithms.
- *Incremental* clustering algorithms, which are based on the assumption that it is possible to consider (and transfer to main memory) the items data one at a time and assign the items to existing clusters.
- *Divide and conquer* approach: The item set is stored in secondary memory (for instance, an hard disk) and is divided in subsets. The subsets are then clustered independently, then a merging step produces the clustering of the entire item set.
- *Parallel* implementations of clustering algorithms.

Initial random sampling is one of the most simple methods and can be virtually applied to any clustering algorithm, but it introduces an approximation in the quality of the results. However, if the sample size is large enough, the result of the data mining method on the sample will be similar enough to the result on the original item set [58]. Among the more advanced data summarization methods available, one of the most interesting is the *data bubble* approach [58], a new method, based only on distance information, that can be applied directly to the items data. Each bubble summarizes the data of entire groups of similar items by means of appropriate values, for instance the centroid or the radius (extent) of the bubble. The clustering algorithm can then be easily adapted to work on the bubbles representing the items instead of the items themselves.

As to the incremental option, many algorithms exist, such as the cobweb system [53] or the shortest spanning path (SSP) algorithm [54]. However, the divide and conquer and parallel options are preferable, since they make it possible again to exploit the usual clustering algorithms for large data sets, as the ones we presented earlier in this section, which provide the well-known high level of effectiveness in such scenarios. For instance, the spherical K-means algorithm and most of its variants have been successfully parallelized [55]. In such algorithms, the data (*i.e.* the distance matrix) can be easily partitioned among the available nodes, *i.e.* the computer processors or workstations. For the centroids' assignment and recom-putation phases there are no particular needs for communication or coordination, therefore these steps of the algorithm can be trivially parallelized via data partitioning. If the chosen algorithm involves the buckshot variant, parallelizing the creation of the initial clusters via hierarchical agglomerative clustering can be harder, but even this problem has been successfully solved [56]: A single similarity matrix is kept consistent among all nodes, requiring communications whenever updates are performed.

Parallel implementations generally offer an efficiency that is directly proportional to the number of involved nodes and an efficacy that is equal to the 'standard' techniques, since they are usually designed to produce the same results as the serial implementations.

3.6.6 Comprehension and navigation of clustering output

As we said in the introduction of this section, a good text clustering technique should produce high quality results which not only maximize mathematical quality functions of inter-cluster similarity and/or intra-cluster dissimilarity, but which are also clearly interpretable by the end user. There are several ways to make the resulting clusters more understandable. For instance, one could exploit a clustering algorithm that does not produce difficult to interpret 'reduced' dimensions by means of standard dimensionality reduction techniques, but which works directly on the original dimensions, automatically identifying easily understandable clusters in appropriate subspaces (see CLIQUE in subsection 3.2.3). Consider this simple example: The scenario is text clustering with documents described by vectors of their features, each feature being the frequency of a particular term in a document. CLIQUE algorithm finds clusters in a 2-dimensional subspace corresponding to two particular features, the terms 'computer' and 'game'. The resulting clusters contain groups of documents in which the frequency of such terms is between a particular range. An example of an outputted cluster C_i could be:

$$C_i = \left\{ Doc \;\middle|\; \left(a_1 < tf_{Doc}^{'computer'} < a_2\right) \wedge \left(b_1 < tf_{Doc}^{'game'} < b_2\right) \right\}$$

where $tf_{Doc}^{'term'}$ is the term frequency of 'term' in document *Doc*. For instance, a cluster could contain documents presenting particularly high frequencies for both terms; such cluster would easily be interpreted as 'documents about computer AND game', *i.e.* 'documents about videogames'.

An even more explicit way of presenting clusters to the user is to associate to them an explicit textual description. Such description should be automatically generated by the clustering technique and, most importantly, should be easily understandable also to non-expert users [57]. For instance, the user of a document search engine could be guided by such descriptions in browsing the presented clustering results. The FTC algorithm (see subsection 3.2.4) describes (and represents) each cluster as a list of frequent terms which are present in all its documents. Such terms are carefully selected by the algorithm and typically represent the main topics of the associated documents. By employing the hierarchical version of FTC, HTFC, the user can easily navigate the output tree from the root (all documents) to a leaf (a particularly restricted topic described by many terms), by reading such descriptions of the clusters and gradually choosing more terms to refine its search (see fig. 7).

An even more sophisticated approach in presenting the clustering results is the interactive one, as offered by the Scatter/Gather clustering and browsing system [44]. Such system uses the K-means algorithm, enhanced with its buckshot and refinement variations, to cluster documents 'on the fly' as the user navigates the document set. Each cluster is represented by a *cluster digest*, which contains, along with the mathematical information required for distance computations, a textual description of the cluster. Such description is composed by the cluster's topical terms, *i.e.* the terms appearing most frequently in the

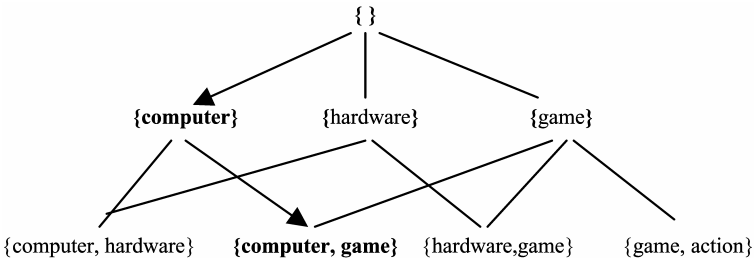


Figure 7: Navigation of a hierarchical output.

group as a whole. The system first presents a restricted set of general topics by *scattering* the document collection in a number of clusters and presenting the descriptions of such clusters to the user (see fig. 8, where, for simplicity, the cluster descriptions have been reduced to the most important term). Then, the user selects one or more of the groups for further study. The system *gathers* together such groups and performs clustering again to scatter the new collection in more particular groups, and so on.

3.7 Clustering web documents

Nowadays, hypertext documents represent a great portion of the available textual data. The World Wide Web (WWW) contains billions of hypertext documents on almost all topics. Moreover, hypertext documents are used for digital libraries, product catalogues, reviews, newsgroups, medical reports, customer service reports and so on. While the users are enjoying such large amounts of available hypertext documents, they pose problems of analysis and summarization. As a consequence, text clustering could not disregard these kind of data; thus, part of the recent

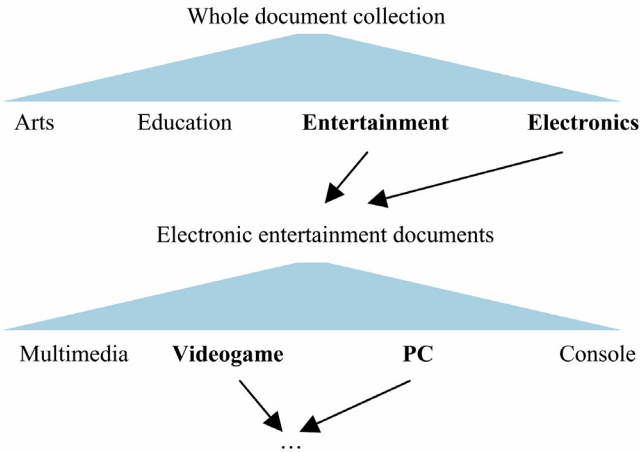


Figure 8: Interactive scatter/gather navigation of clustering output.

efforts in the text clustering field has been devoted to the problem of clustering hypertext documents and, in particular, web documents. Since the beginning, it was evident that traditional text clustering techniques were not an adequate solution to this problem, mainly because clustering web documents relying only on the textual context is not very effective [33]. Indeed, apart from the huge size and high dynamics, the WWW scenario differs significantly from the traditional IR one. The Web is a populist hypermedia relying on a rich structure of both textual web documents and hyperlinks that connect them. There is no consistent style or standard; content is created autonomously by the page authors and hyperlinks are added, for instance, for navigation, endorsement, citation, criticism, plain whim.

Web clustering helps end-users and information retrieval systems to organize such a vast amount of hypertext documents in different ways. *Offline clustering* is used to divide the entire web crawl data into a number of groups to limit the search scope. Moreover, *online clustering* can make the results of search queries easier to browse. Most search engines return large amounts of results also when a simple query is submitted. On the other hand, users, with their limited amount of time, usually process only the first few results. Text clustering can be applied to structure the large result set according to topics, such that it can be interactively browsed by users. Compared to offline clustering, whose requirements are quite similar to those of text clustering, for online clustering some key requirements have to be considered [39]:

Overlap: As documents have multiple topics, each document does not necessarily need to belong to one cluster.

Few information-tolerance: The method ought to produce very high quality clusters even if it does not download the original documents off the web. As most users are unwilling to wait, the method should only access the snippets returned by the search engines and, in case, the topology of the web documents given by the hyperlinks connecting them.

Speed: The clustering method ought to be able to cluster up to one thousand snippets in a few seconds.

Incrementality: To save time, the method should start to process each snippet as soon as it is received over the Web.

Feature selection is a major problem related to text clustering. On the other hand, the traditional term weighting is not suitable in the Web domain [40]. The usual problems of synonymy, polysemy and context sensitivity of IR become especially severe on the Web. Moreover, misrepresentation of content by 'spamming' the pages with meaningless terms is quite frequent. The paper [34] provides a statistical analysis of the word distribution in the Web pages. On a collection of 20,000 Web pages collected from the Yahoo! search engine, they noticed that most of the documents contain few words (less than 500) and that the average term frequency is very low (less than 2.0). In this way, the *tfxidf* weighting scheme may rank those features having

relatively low document frequency highly, as the low document frequency will give an higher weight to the feature. Such features can lead to a clustering result containing many small clusters, which, given the large dimension of the starting web document collections, may not be particularly effective both for offline and online clustering. Last but not least, a new dimension is added because of extant partial structure: Web documents are hypertexts containing hyperlink in addition to text. Previous work in IR on the web has recognized that the hyperlink structure can be very valuable for locating information (see *e.g.* [35, 36]). This assumes that if there is a link from page *v* and *w*, then the author of *v* recommends page *w*, and links often connect related pages. Thus, in the context of clustering web documents, in addition to the textual contents of documents, many other sources of information can be effectively used to enhance clustering effectiveness, such as hyperlinks and co-citation (co-reference) patterns [37] between documents.

Some clustering algorithms quite neglect the page contents and use only the information about the links that appear on each page. This is the case of the Companion and Cocitation algorithms proposed in [38]. They both have been developed as online clustering algorithms, in which, similarly to the 'What's related' algorithm in Netscape, clustering is done on the results of search queries, according to topic. The former exploits links whereas the latter exploits co-citations. In particular, the Companion algorithm is derived from the HITS algorithm proposed by Kleinberg for ranking search engine queries [39]. It takes as input a starting URL *u* and consists of four steps:

1. build a vicinity graph for *u*. It is a directed graph of nodes that are next to *u* in the web graphs. Graph nodes correspond to web pages and graph edges correspond to hyperlinks;
2. contract duplicates and near-duplicates in this graph. This step is devoted to the elimination of near-duplicates, that is pages which are duplicates across hosts (*e.g.* mirror sites, aliases) and which can greatly distort the results;
3. compute edge weights based on host and host connections. In this stage, they assign a weight to each edge, using the edge weighting scheme of HITS [39] which distinguishes between hub and authority weights (see next step);
4. compute the hub score and an authority score for each node in the graph and return the top ranked authority nodes. The algorithm is a straightforward extension of the HITS algorithm with edge weights. The intuition behind the HITS algorithm is that a document that points to many others is a good hub, and a document that many documents point to is a good authority. The HITS computation repeatedly updates hub and authority scores so that documents with high authority scores are expected to have a relevant content, whereas documents with high hub scores are expected to contain links to relevant content. It then returns the nodes with the ten highest authority scores as the pages that are most related to the start page *u*.

The Cocitation algorithm is a variant of the Companion one which examines the siblings of a starting node *u* in the web graph, as two nodes are co-cited if they have a common parent. The number of common parents of two nodes is their degree of co-citation. Such an algorithm determines related pages by looking for sibling nodes with the highest degree of co-citation.

A completely different approach is adopted by those algorithms which perform online clustering by only exploiting the snippets returned by the search engines. Among those, the most cited approach is the Suffix Tree Clustering (STC) proposed in [40]. It is an incremental and linear time algorithm which relies on a suffix tree to efficiently identify sets of documents that share common phrases and uses this information to create clusters and to succinctly summarize their contents to users. In particular, rather than using terms, it represents each document as the set of sentences the corresponding snippet is made up. Then, it identifies the base clusters, *i.e.* sets of documents that share a common phrase, by creating an inverted index of phrases for the document collection. This is done efficiently using a data structure called *suffix tree* [40]. A phrase is an ordered sequence of one or more words and a suffix tree of a set of sentences is a compact trie containing all the suffixes (*i.e.* phrases) of all the sentences in the set. As a first step, from each document returned by the search engine and summarized by its snippet, the STC algorithm extracts the set of the snippet sentences and stems them. Such set of stemmed sentences is then inserted into the suffix tree. In particular:

1. Each internal node of the suffix tree has at least 2 children.
2. Each edge is labelled with a non-empty sub-sentence. The label of a node is the concatenation of the edge-labels on the path from the root to that node.
3. No two edges out of the same node can have edge-labels that begin with the same word.
4. For each suffix, there exists a suffix-node whose label equals the suffix.

Figure 9 shows the suffix tree of a set of stemmed sentences: ‘cat ate cheese’, ‘mouse ate cheese too’, and ‘cat ate mouse too’. The nodes of the suffix are drawn as circles. Each suffix-node has one or more boxes attached to it designating the suffix from which it originates. The first number represents the sentence whereas the second number designates the initial position of the suffix in the sentence.

Notice that each node of the suffix tree represents a group of documents (the set of documents tagging the suffix-nodes descendant of the node) and the label of the node is the phrase that is common to all of them. Therefore, each node represents a base cluster. For instance, in fig. 9 there are 6 base clusters marked with the small letters *a-f* and node *a* is the set of documents {1,3} sharing the phrase ‘cat ate’. Each base cluster is thus assigned a score that is a function of the number of documents it contains, and the words that make up its phrase. Documents may share more than one phrase. As a result, the document sets of distinct base clusters may overlap and may even be identical.

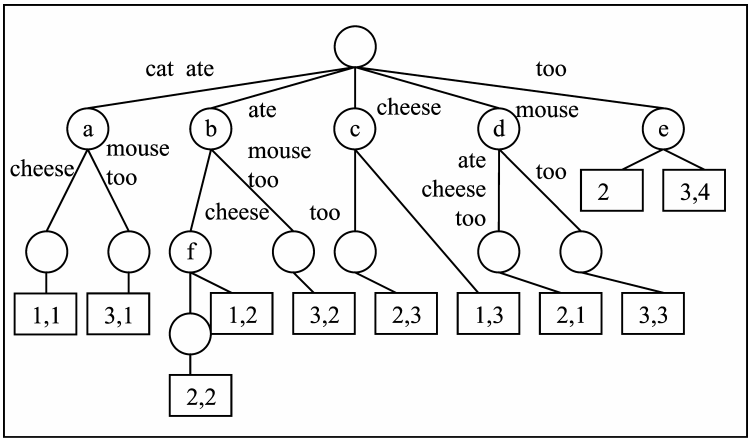


Figure 9: Suffix tree example.

After having identified the base clusters, the STC algorithm merges base clusters with a high overlap in their document sets. To this end, a binary similarity measure is defined between base clusters based on the overlap of their document sets. In particular the similarity between two base clusters B_1 and B_2 is defined to be 1 if the number of documents common to the two clusters $|B_1 \cap B_2|$ is greater than half of the number of the documents of one of the two clusters, *i.e.* $0.5\ |B_1|(|B_2|)$, 0 otherwise. The final clusters are obtained by merging such base clusters having a similarity of 1. They are scored and sorted based on the scores of their base clusters and their overlap. As the number of clusters may vary, they report only the top few ones. As a final remark, notice that the STC algorithm meets all the online clustering key requirements.

Hybrid approaches comprehensively incorporate information from hyperlink structures, co-citation patterns and textual contents to construct similarity measures for weighting the homogeneity of web documents. For instance, it is possible to define a similarity matrix by combining an hyperlink matrix with a textual similarity matrix and a co-citation matrix in the following way. Given the link graph $G=(V,E)$ of the set of documents where V is the set of documents and E is the set of links between the documents, the hyperlink matrix $A=(a_{ij})$ is defined as the adjacency matrix of the graph where each element a_{ij} is 1 if an edge between the vertexes i and j exists, 0 otherwise. The textual similarity matrix S can be constructed by utilizing the entire text of each web document and by measuring the similarity between two web documents by means of the vector space model. Finally, a co-citation matrix $C=(c_{ij})$ can be defined such that c_{ij} is the number of web documents pointing to both i and j. Such different kinds of knowledge have been combined in [36] in the similarity matrix *Sym* defined as

$$Sym = \alpha \frac{A \otimes S}{\|A \otimes S\|} + (1-\alpha)\frac{C}{\|C\|}$$

where $\otimes$ means multiplication element by element and α is a real number between 0 and 1. Specifically, the hyperlink structure is used as the dominant factor in the similarity metric, and the textual contents are used to modulate the strength of each hyperlink. The similarity matrix can thus be used as a basis for any of the clustering algorithms proposed for text clustering. In [37] the offline clustering problem is defined as a problem of partitioning the link graph into connected sub-graphs. It is solved by a spectral graph partitioning method which uses Fieldler vector of the similarity matrix *Sym* to separate clusters, measured by normalized cut criterion. In this chapter we do not describe further this approach and refer interested readers to [37]. Other different approaches for offline clustering represent documents by only considering specific information in the contents. For instance, in [42] the authors define the bag representing a document to be a multiset of occurrences of words near a hyperlink to the page. Indeed, when pages are linked to, the anchor text, as well as the text surrounding the link (anchor-window) are often succinct descriptions of the page [43]. In this way, the detrimental effects of synonymy are reduced, since the union of all anchor-windows will likely contain most variations of words strongly indicative of the target page's content. In [34], instead, a feature extracting method is proposed which balances the trade-off between the coverage and the number of features used for document representation. The coverage of the features is defined as the percentage of documents containing at least one of the extracted features and if the coverage is too low, there will be many documents represented by a feature vector with zero weight in all entries.

4 Conclusions

In this chapter we have covered selected topics on textual cluster analysis in a mining scenario. In our discussion, we tried to be as exhaustive on the selected aspects as possible and to cover the most important solutions to the main problems related to such a theme. However, since text clustering is an incredibly large and multifaceted subject, it would have been impossible to deeply analyze every detail of such a large world. Therefore, we advise readers to make use of the given references in order to obtain further details about the topics of her/his interest. Further, as we already said in the introduction, text clustering is a very dynamic field in constant evolution. We described the most up-to-date text clustering solutions, nonetheless readers should expect and look for many other interesting solutions to be proposed in the future.

References

[1] Fu, K.S. & Mui, J.K., A survey on image segmentation. *Pattern Recognition*, **13(1)**, pp. 3–16, 1981.

[2] Dorai, C. & Jain, A.K., Shape spectra based view grouping for free form objects. *Proc. of the Int. Conf. on Image Processing* (ICIP-95), pp. 240–243, 1995.

[3] Frakes, W.B. & Baeza-Yates, R., (eds), *Information Retrieval – Data Structures and Algorithms*, Prentice Hall: Englewood Cliffs, 1992.

[4] Chen, M-S, Han, J. & Yu, P.S., Data mining: An overview from a database perspective. *IEEE Trans. on Knowledge and Data Eng.* (TKDE), **8(6)**, pp. 866–883, 1996.
[5] Anderberg, M.R, *Cluster analysis for applications*, Academic Press: New York, 1973.
[6] Hartigan, J.A., *Cluster algorithms*, Wiley: New York, 1975.
[7] Everitt, B., *Cluster analysis*, 2nd ed., Halsted: New York, 1980.
[8] Kaufman, L., Finding groups in data: An introduction to cluster analysis, Wiley: New York, 1990.
[9] Jain, A.K., Murty, M.N. & Flynn, P.J., Data clustering: A review. *ACM Computing Surveys*, **31(3)**, pp. 264–323, 1999.
[10] Salton, G., *Automatic Text Processing,* Addison-Wesley: Reading, 1989.
[11] Mao, J. & Jain, A.K., A self-organizing network for hyperellipsoidal clustering (HEC). *IEEE Transactions on Neural Networks,* **7(1)**, pp. 16–29, 1996.
[12] Salton, G., *The SMART Retrieval System*, Prentice Hall: Englewood Cliffs, 1971.
[13] Rasmussen, E., Clustering algorithms (Chapter 16). In [3], pp. 419–442.
[14] Sneath, P.H.A. & Sokal, R. R., *Numerical Taxonomy,* Freeman: London, 1973.
[15] King, B., Step-wise clustering procedures. *Journal of the American Statistical Association,* **69**, pp. 86–101, 1967.
[16] Baeza-Yates, R., Introduction to data structures and algorithms related to information retrieval (Chapter 2). In [3], pp. 13–27.
[17] Dubes, R.C., How many clusters are best?—an experiment. *Pattern Recogn.*, **20(6)**, pp. 645–663, 1987.
[18] Steinbach M., Karypis G. & Kumar V., A comparison of document clustering techniques. *KDD Workshop on Text Mining*, 2000.
[19] Day, W.H.E., Complexity theory: An introduction for practitioners of classification. *Clustering and Classification*, eds. P. Arabie & L. Hubert, World Scientific Publishing Co. Inc.: River Edge, 1992.
[20] Pirolli, P., Schank, P., Hearst, M. & Diehl, C., Scatter/gather browsing communicates the topic structure of a very large text collection. *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 1996.
[21] Dhillon, I.S. & Modha, D.S., Concept decomposition for large sparse text data using clustering. *Machine Learning*, **42(1)**, pp. 143–175, 2001.
[22] Agrawal, R., Gehrke, J., Gunopulos, D. & Raghavan, P., Automatic subspace clustering of high dimensional data for data mining applications. *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 94–105, 1996.
[23] Yang, Y. & Pederson, J.O., A comparative study on feature selection in text categorization. *Proc. of the 14th International Conference on Machine Learning (ICML'97)*, pp. 412–420, 1997.
[24] Dhillon, I., Kogan, J. & Nicholas, C., Feature selection and document clustering. *Text Data Mining and Applications*, 2002.
[25] Guyon, I. & Elisseeff, A., (eds), Special issue on variable and feature selection, *Journal of Machine Learning*, **3(mar)**, 2003.

[26] Salton, G. & McGill, M.J., *Introduction to modern information retrieval*, McGraw-Hill: New York, 1983.
[27] Wilbur, J.W. & Sirotkin, K., The automatic identification of stop words, *Journal of Information Science*, **18(1)**, pp. 45–55, 1992.
[28] Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W. & Harshman, R.A., *Indexing by latent semantic analysis. Journal of the Society for Information Science*, **41(6)**, pp. 391–407, 1990.
[29] Frankl, P. & Maehara, H., The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory*, **B(44)**, pp. 355–362, 1988.
[30] Papadimitriou, C.H., Raghavan, P., Tamaki, H. & Vempala, S:, Latent Semantic Indexing: A Probabilistic Analysis. *Journal of Computer and System Sciences*, **61(2)**, pp. 217–235, 2000.
[31] Chakrabarti, S., Data mining for hypertext: A tutorial survey. *SIGKDD Explorations*, **1(2)**, pp. 1–11, 2000.
[32] Schütze, H. & Silverstein, C., Projections for efficient document clustering. *Proc. of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 74–81, 1997.
[33] Willett, P., Recent trends in hierarchical document clustering. *Information Processing and Management*, **24(5)**, pp. 577–597, 1988.
[34] Wong, W. & Fu, A.W., Incremental document clustering for web page classification. *Proc. of the IEEE International Conference on Information Society in the 21st century*: *emerging technologies and new challenges*, 2000.
[35] Arocena, G.O., Mendelzon, A.O. & Mihaila, G.A., Applications of a web query language. *Proc. of the 6th International World Wide Web Conference*, pp. 587–595, 1997.
[36] Bharat, K. & Henzinger, M., Improved algorithms for topic distillation in hyperlinked environments. *Proc. of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 104–111, 1998.
[37] He, X., Zha, H. Ding, C.H.Q. Simon, H.D., Automatic Topic Identification Using Webpage Clustering, *Proc. of the IEEE International Conference on Data Mining (ICDM)*, pp. 195–202, 2001.
[38] Dean, J. & Henzinger, M.R., Finding related pages in the world wide web. *Proc. of the 8th International World Wide Web Conference*, pp. 389–401, 1999.
[39] Kleinberg, J., Authoritative sources in a hyperlinked environment. *Proc. of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pp. 668–677, 1998.
[40] Zamir, O. & Etzioni, O., Web document clustering: A feasibility demonstrator. *Proc. of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 46–54, 1998.
[41] Ukkonen, E., On-line construction of suffix trees. *Algorithmica*, **14(2)**, pp. 49–60, 1995.
[42] Haveliwala, T.H., Gionis, A. & Indyk, P., Scalable techniques for clustering the web, *Proc. of the 3rd International Workshop on the Web and Databases*, pp. 129–134, 2000.

[43] Amitay, E., Using common hypertext links to identify the best phrasal description of target web documents, *Proc. of the SIGIR Workshop on Hypertext Information Retrieval for the Web*, 1998.

[44] Cutting, D.R., Pedersen, J.O., Karger, D.R. & Tukey, J.W., Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* eds. N.J. Belkin, P. Ingwersen & A.M. Pejtersen, ACM Press: New York, pp. 318–329, 1992.

[45] Larsen, B. & Aone, C., Fast and Effective Text Mining Using Linear-Time Document Clustering. *Proc. of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 16–22, 1999.

[46] Dhillon, I., Guan, Y. & Kogan, J., Refining clusters in high dimensional data. *Proc. of Second SIAM ICDM Workshop on clustering high dimensional data*, Arlington, 2002.

[47] Faber V., Clustering and the Continuous k-Means Algorithm. *Los Alamos Science Magazine*, **22**, pp. 138–144, 1994.

[48] Michaud, P., Clustering Algorithms. *Future Generation Computer Systems*, **13(2-3)**, pp. 135–147, 1997.

[49] Marcotorchino, J.F. & Michaud, P., Heuristic approach of the similarity aggregation problem. *Methods Oper. Res.*, **43**, pp. 395–404, 1981.

[50] Michaud, P., Simulated computation in automatic classification. *Proc. 2nd Symposium on High Performance Computing*, eds. M. Durand & F. El Dabaghi, pp. 381–396, 1991.

[51] Ng, R.T. & Han, J., Efficient and Effective Clustering Methods for Spatial Data Mining. *Proc. of 20th International Conference on Very Large Data Bases*, eds. J. Bocca, M. Jarke & C. Zaniolo, Morgan Kaufmann Publishers: Los Altos, pp. 12–15, 1994.

[52] Zhang, T., Ramakrishnan, R., Livny, M., BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data*, eds. H.V. Jagadish & I.S. Mumick, ACM Press, pp. 103–114, 1996.

[53] Fisher, D., Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, **2**, pp. 139–172, 1987.

[54] Slagle, J.R., Chang, C.L. & Heller, S.R., A clustering and data-reorganizing algorithm. *Proc. of IEEE Transaction Systems Man and Cybernetics*, **5**, pp. 125–128, 1975.

[55] Dhillon, I., Fan, J. & Guan, Y., Efficient Clustering of Very Large Document Collections. *Data Mining for Scientific and Engineering Applications*, eds. R. Grossman, G. Kamath & R. Naburu, Kluwer Academic Publishers, 2001.

[56] Jensen, E.C., Beitzel, S.M., Pilotto, A.J., Goharian, N. & Frieder, O., Parallelizing the buckshot algorithm for efficient document clustering. *Proc. of the eleventh international conference on Information and knowledge management*, ACM Press: New York, pp. 684–686, 2002.

[57] Beil, F., Ester, M. & Xu., X., Frequent term-based text clustering. *Proc. of the 8th International Conference on Knowledge Discovery and Data Mining (KDD 02)*, ACM Press: New York, pp. 436–442, 2002.

[58] Zhou, J. & Sander, J.: Data Bubbles for Non-Vector Data: Speeding-up Hierarchical Clustering in Arbitrary Metric Spaces. *Proc. of 29th International Conference on Very Large Data Bases,* Kaufmann, pp. 452–463, 2003.

[59] Zaït, M. & Messatfa, H., A comparative study of clustering methods. *Future Generation Computer Systems*, **13**, pp. 149–159, 1997.

[60] Dörre, J., Gerstl, P. & Seiffert, R., Text Mining: Finding Nuggets in Mountains of Textual Data. *Proc. of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press: New York, pp. 398–401, 1999.