A Conceptual Clustering Approach for User Profiling in Personal Information Agents

Daniela Godoy and Analía Amandi

ISISTAN Research Institute
Univ. Nacional del Centro de la Prov. de Bs. As.
Also at CONICET
Campus Universitario, CP 7000
Tandil, Bs. As., Argentina

1. Introduction

Information agents personalizing information related tasks based on the content of documents and the user interests offer users an alternative to cope with the vast amount of information available on the Web. For this reason, they have been largely applied in the last years to accomplish several tasks, such as searching, filtering and summarizing information on behalf of a user or community of users [21].

For agents personalizing information related tasks, the capacity of acquiring and modeling user interest categories into user profiles becomes a critical aspect in agent design. The knowledge contained into profiles allows agents to categorize documents based on the features they exhibit, even though it may not be possible for them to assess the full meaning of documents. In consequence, agent effectiveness depends on both profile accuracy and completeness.

In the user profiling task, document clustering offers the advantage that an a priori knowledge of categories is not needed, so that agents can deal with unpredictable interest domains by learning the user interests in a completely unsupervised process. However, clustering approaches to be applied in this context are constrained by the characteristics of personal agents. These agents not only have to summarize categories corresponding to diverse user information interests into a profile (e.g. interests related to their hobbies, like sports or movies, or to their work, like politics), but also have to describe user experiences in each of these categories at different levels of abstraction. For instance, a user interest in skydiving may respond to a more general interest in winter sports, or even in just sports. In addition, since personal agents interact with the environment and the user over time, the complete learning process should be incremental.

Keywords: conceptual clustering, user profiling, personal information agents
The first requirement imposes a constraint on the representation and organization of knowledge, suggesting a hierarchical organization of user interest categories. The second, in turn, on the mechanisms to support learning, as agents have to be able to incrementally discovering interest categories. In order to meet both requirements we propose a document clustering algorithm belonging to the conceptual clustering paradigm, named WebDCC (Web Document Conceptual Clustering) [11], designed to support learning over personal information agents. WebDCC carries out incremental, unsupervised concept learning over Web documents in order to allow agents to build and maintain user profiles. Like other conceptual clustering algorithms, this algorithm not only performs clustering, but also characterization, i.e., the formation of intentional concept descriptions for extensionally defined clusters. Hence, the algorithm allows agents to acquire hierarchical descriptions of user interests. The assessing of such profiles enables agents to generate comprehensible representations of user interests to be presented to other users, helping to establish trust in agent recommendations, or other agents, making it possible to establish a collaborative behavior in an agent community.

This paper is structured as follows. Section 2 introduces the WebDCC algorithm and its principal components. Empirical evaluation of the algorithm is summarized in Section 3. Section 4 places this work in the context of related ones. Finally, concluding remarks and future lines of research are described in Section 5.

2. WebDCC Overview

WebDCC is an algorithm belonging to the conceptual clustering paradigm that carries out incremental, unsupervised concept learning over Web documents. First introduced by [20], conceptual clustering include not only clustering, but also characterization, i.e., the formation of intentional concept descriptions for extensionally defined clusters. More formally, conceptual clustering is defined as the task of, given a sequential presentation of instances and their associated descriptions, finding clusters that group these instances into concepts or categories, a summary description of each concept and a hierarchical organization of them [35].

Experiences representing user interests that agents can capture through observation, such as Web pages the user browse or pieces of news the user read in an on-line newspaper, are incrementally presented to WebDCC algorithm, which is concerned with forming hierarchies of concepts starting from them. Thus, instances in this algorithm correspond to vector representations of Web pages according to the vector space model described in [32]. In this model, each document is identified by a feature vector in a space in which each dimension corresponds to a distinct term associated with a numerical value or weight which indicates its importance. The resulting representation of a Web page is, therefore, equivalent to a t-dimensional vector:

\[ d_i = (t_1, w_1, ..., t_t, w_t) \]

where \( w_j \) represent the weight of the term \( t_j \) in the instance or document \( d_i \). Before obtaining a document representation, non-informative words such as prepositions, conjunctions, pronouns and very common verbs are removed by using a standard stop-word list. A stemming algorithm is applied to the remaining words in order to reduce the variant forms of a word to a common one. In the experiments reported in this paper, terms were stemmed using the Porter stemming algorithm [28].

Instances obtained by extracting feature vectors from documents are incrementally presented to the WebDCC algorithm, which is concerned with building a hierarchy of concepts. Because of its incremental nature, the algorithm requires a sequential presentation and processing of instances which, even when stored, are not extensively reprocessed in further stages. Based on instances presented to the algorithm in an on-line fashion, it outputs a concept hierarchy describing the categories the instances belong to.

Hierarchies of concepts produced by WebDCC algorithm are classification trees in which internal nodes represent concepts and leaf nodes represent clusters of instances. The root of the hierarchy corresponds to the most general concept, which comprises all the instances the algorithm has seen, whereas inner concepts become increasingly specific as they are placed lower in the hierarchy, covering only subsets of instances by themselves. In-
In other words, a hierarchy is formed by an arbitrary number of concepts, denoted by $C = \{c_1, c_2, ..., c_n\}$, which are gradually discovered by the algorithm as new instances become available. In order to automatically assign instances to concepts, a text-like description given by a set of weighted terms, denoted $c_i = \{(t_1, w_1), ..., (t_m, w_m)\}$, is associated to concepts during the process of hierarchical concept formation. Notice that any subset of instances is considered a category, while a concept is the internal representation of the category. For example, suppose the algorithm has discovered the concept $sports$ within a group of instances. In this case, the concept itself consists in a description such as $c_{sports} = \{(sports, 0.5), (score, 0.3), (team, 0.2)\}$, which summarizes the category $sports$ composed of by all instances the algorithm has seen about $sports$. Then, the algorithm is capable of creating and deleting categories, classifying instances into categories and manipulating the corresponding concepts by creating, updating or deleting them.

Leaves in the hierarchy correspond to clusters of instances belonging to all the ancestor concepts. Intuitively, clusters correspond to groups of instances whose members are more similar to one another than to members of other clusters, so that clusters group highly similar instances observed by the algorithm. In general terms, a set of $n_i$ instances or documents belonging to a concept $c_i$ and denoted $D_i = \{d_1, d_2, ..., d_{n_i}\}$, are organized into a collection of $k$ clusters below $c_i$, $S_{ij} = \{s_{1j}, s_{2j}, ..., s_{kj}\}$, containing elements of $D_i$ such that $s_{ij} \cap s_{pj} = \emptyset$, $\forall l \neq p$.

Figure 1 shows an example of a conceptual hierarchy which is possible to achieve with WebDCC algorithm. In this hierarchy most instances belong to two broad categories, $sports$ and $politics$, represented by the concepts $c_{politics} = \{(politics, 0.3), (president, 0.2), (election, 0.2)\}$ and $c_{sports} = \{(sports, 0.5), (score, 0.3), (team, 0.2)\}$ respectively. Instances belonging to other categories, for instance documents about $finances$, are placed in either categories or clusters at the same level, but outside $politics$ and $sports$. Each concept has a number of child concepts covering different aspects of their parent concepts. Hence, in a more specific level in the hierarchy it is possible to find two sub-concepts of $sports$, $basketball$ and $football$, described by their own set of terms, $c_{basketball} = \{(basketball, 0.6), (NBA, 0.5)\}$ and $c_{football} = \{(football, 0.3), (FIFA, 0.2)\}$, and no sub-concepts of $politics$. Since they have no further children, the concepts $c_{basketball}$, $c_{football}$ and $c_{politics}$ are terminal nodes in the hierarchy. Instances placed below any of these concepts, necessarily belong to both the terminal concept and all its ancestors. Leaf nodes in the hierarchy are clusters of instances grouped by similarity based on their own attributes. In the example, instances in the $football$ category are organized into a single cluster, whereas instances in the $basketball$ category are partitioned into two different clusters. If the algorithm continues starting from this point, two new concepts distinguishing the two different sort of instances about $basketball$ will possibly emerge.

WebDCC algorithm takes as input a hierarchy $H$ and a new instance $d_{new}$, and returns an updated hierarchy $H'$ resulting from the addition of $d_{new}$ to $H$. Figure 2 depicts the main stages of the clustering process. Initially, instances are classified at the current hierarchy of concepts which at the beginning is compose only by the root. Then, incremental clustering is performed below the concepts instances belong to in order to create a flat partition of instances. The incorporation of instances to the hierarchy is followed by an evaluation of the current hierarchical structure in order to determine whether new concepts can be created or some restructuring is needed. If this
is the case, a hierarchical concept formation stage extracts meaningful concepts to refine the hierarchy and reorganize previously discovered concepts. The different stages of WebDCC algorithm are explained in the following subsections.

2.1. Hierarchical Classification Approach

In the first stage of WebDCC algorithm, instances are assigned to categories at the current hierarchy of concepts. At any moment, this hierarchy could be formed by a single concept representing the root category or several categories in one or more hierarchical levels. In other words, a text categorization problem is stated in which a set of documents \( D = \{d_1, d_2, \ldots, d_n\} \) have to be assigned to a set of predefined categories \( C = \{c_1, c_2, \ldots, c_k\} \). Particularly, since categories are arranged in a hierarchical fashion, the task of categorizing text is known as hierarchical text categorization (HTC) as opposed to flat text categorization (FTC).

Text categorization into hierarchies or taxonomies is a particular type of multi-label classification problem in which documents belonging to a category in the hierarchy also belong to all its parent categories. In consequence, the categories a document belongs to can be subsequently determined at each level going downward in the hierarchy. Namely, once having selected a category at a certain level, only its children should be considered as prospective categories at the next level.

WebDCC gradually trains classifiers to recognize constitutive features of categories and thus discriminate documents belonging to these categories. For each node in the hierarchical structure, a separate classifier is induced to distinguish documents that should be assigned to the category it represents from other documents. Thus, at each decision point in the hierarchy a classifier is concerned with a binary classification problem where a document is either relevant or not to a given category.

The classification task proceeds in a top-down manner based on the hierarchical organization of classifiers. Initially, an instance belonging to the root category (all instances) is categorized by classifiers at the first level. Then, classifiers at the second level take the instance that has been classified into some category in the previous level and classify it into categories at the second one. This procedure continues until the instance has reached some leaf node in the hierarchy or it can not be further classified down.

By exploiting the hierarchical relationships between categories, the global classification problem is divided into a set of smaller problems corresponding to the splits in the hierarchy. The basic assumption of hierarchical text categorization is that this decomposition leads to more accurate, specialized classifiers. In addition, it is possible to use a much smaller set of features for each of these subproblems. Indeed, a classifier categorizing documents into a given category only requires to be focused on a small number of features, those features which best discriminate documents belonging to the category from other documents. For example, it seems to be a fairly small set of terms, such as sports, score and team, whose presence in a document clearly determines whether a document belongs to the sports category. However, these terms are unlikely to be useful for either a classifier at the same hierarchical level (e.g. politics) or a classifier at the next hierarchical level (e.g. basketball). In the first case, complete different sets of terms describing the sibling categories would be more useful (e.g. politics, president and election for politics category), whereas in the second case more specific features would be more valuable (e.g. basketball and NBA). Thus, although each classifier uses only a small set of features, the overall set of features is used at different points of the classification process.

Many works have combined hierarchical text categorization with feature subset selection to improve classification accuracy and reduce computational overhead by finding the best subset of features in each classification step. Koller et al. [15] proposed a method for term selection and classifier induction using a Bayesian classifier that allows dependencies between terms. Experiments on small subsets of Reuters\(^1\) collection of newswire articles showed that hierarchical classifiers outperform flat ones when the number of features is small. In [19], naïve Bayes classifiers were combined with feature subset selection of n-grams, whereas a method based on hierarchically structured set of gating and expert networks is presented in [29]. Dumais et al. [6] performed a comparative study aimed at assessing whether taking into account the hierar-

\(^1\)http://www.davidlewis.com/resources/textcollections/
chical structure of categories is beneficial in terms of categorization efficiency and effectiveness. Experiments using Support Vector Machines (SVM) resulted in small advantages of hierarchical classifiers compared with flat classifiers. Mladenic et al. [24] performed a study of feature selection for classification based on text hierarchies. The best performance was achieved by the feature selection method based on Odds ratio, a feature scoring measure known from information retrieval area, and using a relatively small number of features. However, also measures that favor common features such as term frequency obtained good results in the experiments. In all referred works, hierarchical classifiers showed superior performance to flat ones.

The induction of a classifier for a given category \( c_i \in C \) takes two phases [33]. In a first phase a function \( CSV_i : d \rightarrow [0, 1] \) is defined such as for a given document \( d \), it returns a categorization status value, i.e. a number between 0 and 1 that represents the evidence for the fact that \( d \) should be classified under the category \( c_i \). The CSV function takes up different meanings according to the different types of classifiers. In the case of naïve Bayes \( CSV_i(d) \) is a probability, whereas for Rocchio classifiers \( CSV_i(d) \) is a distance between vectors in a \( t \)-dimensional space and so forth. In a second phase, a threshold \( \tau_i \) is defined such that \( CSV_i(d) \geq \tau_i \) is interpreted as the decision to categorize \( d \) under \( c_i \), while \( CSV_i(d) < \tau_i \) is interpreted as the decision not to categorize \( d \) under \( c_i \).

Linear classifiers [17] have a number of interesting properties to be applied in the context of WebDCC algorithm. These classifiers embodies an explicit or declarative representation of the category based on which categorization decisions are taken. Learning of linear classifiers consists in examining the training instances a finite number of times to construct a profile or prototype instance which is later compared against the instances to be classified. Then, a prototype \( p_{c_i} \) for a category \( c_i \) consist in a vector of weighted terms:

\[
p_{c_i} = \{(t_1, w_{11}), ..., (t_p, w_{p1})\}
\]

where \( w_j \) is the weight associated to the term \( t_j \) in the category. The extraction of prototypes is generally preceded by local term space reduction, which consists in selecting the most important terms for the category \( c_i \) according to some evaluation measure.

This kind of classifiers are both efficient, since classification is linear on the number of terms, documents and categories, and easy to interpret, since it is assumed that terms with higher weights are better predictors for the category than those with lower weights. In addition, the representation of linear classifiers is similar to the representation of documents, so that they can take advantage of standard information retrieval techniques.

WebDCC builds a hierarchical set of classifiers, each based on its own set of relevant features, as a combined result of a feature selection algorithm for deciding on the appropriate set of terms at each node in the tree and a supervised learning algorithm for constructing a classifier for such node. In particular, an instantiation of Rocchio classifier is used to train classifiers with \( \beta = 1 \) and \( \gamma = 0 \) since no negative examples are available. In addition, in the context of text classification \( \alpha \) is equal to zero since there is no initial query. The same parameter setting is used in [7], yielding:

\[
p_{c_i} = \frac{1}{|c_i|} \sum_{d \in c_i} d
\]

as a prototype for each class \( c_i \in C \). Hence, each prototype \( p_{c_i} \) is the plain average of all training
instances belonging to the class $c_i$ and the weight of each term is simply the average of its weights in positive instances of the category. The induction of classifiers for hierarchical categories is further explained in Section 2.3.

In contrast to other linear classifiers, this instantiation of Rocchio not only supports incremental insertion, but also remotion of instances without rebuilding the classifiers. This is convenient for user profiling, in which profile adaptation is concerned with both the incorporation of new experiences describing user interests as well as the forgetting of old, no longer interesting ones.

To categorize a new instance into a given category, its closeness to the prototype vector of the category is computed by using the cosine similarity in Equation 4. In turn, the prototype vectors of categories are attached with the classification threshold $\tau$, which indicates the minimum similarity to the prototype of each category instances should have in order to fall into these categories.

The weighted terms associated to categories at each level in the hierarchy act as classifiers so that the categorization process itself iterates over these levels and, concurrently, updates the term weights when a new instance is included in a category to guarantee that each classifier summarizes all instances below it.

Beginning at the highest level, an instance $d_{new}$ is compared with the prototypes for categories at each level. The instance similarity to the prototype vector of a category represents the confidence that the given instance belong to this category. Therefore, if this value is higher than the classification threshold $\tau$ for a certain category, $d_{new}$ is assigned to the category; otherwise, it is considered as not belonging to the category. After comparing $d_{new}$ with all prototypes at a certain hierarchical level three situations are possible:

- that the instance exceeds the threshold for only one category, in which case it is classified in such category and classification continues at the next hierarchical level,
- that the instance exceeds the threshold for more than one category, in which case the instance is classified into the best category and classification continues at the next hierarchical level,
- that the instance does not exceed the threshold for any category, in which case the classification stops and the instance is left at the current hierarchical level.

The motivation behind the hierarchical classification approach is that, since each classifier deals with a smaller and simpler classification problem, each classifier should be able to categorize documents more accurately than if applied to the whole set of categories. However, each instance must go through a number of decision points before reaching a leaf node, with each decision point increasing the likelihood of misclassification. The accuracy achieved by the hierarchical organization of classifiers depends on which of these two effects dominate [3]. In this regard, McCallum [19] criticized the greedy topic selection method because it requires high accuracy at internal (non-leaf) nodes, but WebDCC controls the selection of the best category by using the classification threshold to reduce the risk of misclassification.

2.2. Non-Hierarchical Clustering Approach

At the bottom levels of the hierarchy the algorithm performs a centroid-based clustering. Given the cluster $s_{ji}$ belonging to the category $c_i$, which is composed of a set of documents and their corresponding vector representations, the composite vector $m_{s_{ji}}$ is defined as:

$$m_{s_{ji}} = \sum_{d \in s_{ji}} d$$  \hspace{1cm} (1)

and the centroid vector $p_{s_{ji}}$ is

$$p_{s_{ji}} = \frac{1}{|s_{ji}|} \sum_{d \in s_{ji}} d$$  \hspace{1cm} (2)

Thus, the centroid $p_{s_{ji}}$ of a cluster $s_{ji}$ is the average point in the multidimensional space defined by the cluster dimensions. The composite vectors of clusters and categories are maintained in WebDCC not to recalculate the centroids every time new instances are added or removed from a category.

When an instance has reached a given concept in the hierarchy, either because it is a terminal node or because it can not be further classified down, it is placed in the most similar cluster below this concept. In order to predict which this cluster is, the closest centroid is determined by comparing the new instance $d_{new}$ with all centroids in the existing clusters. In the case of documents, the distance measure determines the degree of resemblance between the vector representations of both
documents and is frequently calculated by the cosine similarity. The cosine of the angle conformed by two vectors in the space is calculated as the normalized dot product formulated as follows [31]:

\[
sim(d_i, d_j) = \frac{d_i \cdot d_j}{\|d_i\| \|d_j\|}
\]

(3)

\[
= \frac{\sum_{k=1}^{r} w_{ik} \cdot w_{jk}}{\sqrt{\sum_{k=1}^{r} w_{ik}^2} \sqrt{\sum_{k=1}^{r} w_{jk}^2}}
\]

(4)

where \(d_i\) and \(d_j\) are the respective instances, \(w_{ik}\) and \(w_{jk}\) the weights of the word \(k\) in each instance and \(r\) the number of different words in both instances. Like the similarity between documents, the similarity between a centroid vector and an instance is also computed based on the cosine measure.

As the result of instance comparison, the instance \(d_{new}\) is assigned to the cluster with the closest centroid below the category \(c_i\), i.e.

\[
\arg \max_{j=1...k} \sim (d_{new}, p_{s_{ji}})
\]

provided that the similarity is higher than a minimum similarity threshold \(\delta\). Instances not similar enough to any existing centroid according to this threshold cause the creation of new singleton clusters.

The centroid-based approach determines the promising clusters for a new instance using the similarity to centroids, consequently both the similarity measure and, particularly, the similarity threshold have a critical effect on the approach performance. In order to empirically determine a suitable value for the similarity threshold \(\delta\), we carried out a number of experiments over several Web page collections.

In these experiments, the analysis was focused on two aspects of clustering solutions: the threshold \(\delta\) impacts on the total number of clusters and their homogeneity. A value of \(\delta\) near to 0 will produce a very few clusters (only one cluster when \(\delta = 0\)) grouping even highly dissimilar instances, which is almost as informative as the original collection; while a value of \(\delta\) near to 1 will produce a big number of clusters (equal number of cluster that instances when \(\delta = 1\), except for documents with exactly the same representation), which is highly informative but practically useless.

In consequence, clustering solutions are analyzed under variations of \(\delta\) based on the number of resultant clusters and a measure of cluster homogeneity such as Entropy [34]. Labeled training data were used for entropy calculation but not during clustering, expecting clusters generated by the algorithm to be as homogeneous as possible with respect to the labeling of their containing instances. Since the best entropy is achieved when each cluster contains exactly one single instance, both the number of clusters and the cluster homogeneity needs to be analyzed conjunctively.
Experiments were run over two flat Web page collections, such as Syskill/Webert³ and WebType⁴, and a hierarchical collection such as PDDP K-series⁴. Figure 3(a) plots the values of entropy across all possible values of the similarity threshold δ in the three collections. In all cases entropy behaves as expected, decreasing as the number of clusters increases until getting a value of zero when the maximum number of possible clusters is achieved. At that point, each cluster contains a single instance and entropy value is zero. Figure 3(b) shows the increasing in the proportion of clusters \( \frac{s_{\text{clusters}}}{\text{instances}} \) as δ grows. As more relevant documents are included in the clustering solution, more clusters are required to represent the concepts exemplified by these documents. Taken together, Figure 3(a) and (b) demonstrate how the parameter δ allows the WebDCC algorithm to trade off entropy for number of clusters.

The information provided by these two measures is summarized into a single value by the information-theoretic external cluster-validity measure proposed by [5], simply validity measure hereafter. This measure refines entropy to penalize both highly heterogeneous clusters as well as large number of clusters. Figures 4 plots scores achieved with the three collections according to validity measure. These results summarize how the algorithm runs with ten different order of instances.

The lowest values of validity measure were obtained with \( \delta = 0.15 \) in all the collections except from the Syskill/Webert collection in which the lowest validity value was found with \( \delta = 0.1 \). It can be concluded from these results and the validity curves from Figure 4 that the optimal value for the similarity threshold, in which the best relation entropy-number of clusters is attained, is likely to be in the interval 0.1 ≤ \( \delta \) ≤ 0.2. In further experiments we set \( \delta = 0.15 \) as it was the most frequent optimal value in the experiments with the mentioned collections.

2.3. Hierarchical Concept Formation Approach

Hierarchical concept formation takes places as instances are sorted down through the hierarchy.

\[ \text{Validity} = \frac{\text{Valid instances}}{\text{Total instances}} \]

As this occurs, the algorithm not only updates each node as the instances go through in order to guarantee that each concept summarizes all instances below it, but also alters the hierarchical structure after the instance incorporation if necessary. In the last situation, plausible modifications to the hierarchical structure include to extend the hierarchy downward, to create a new disjunct category, to promote, merge, and split categories.

The extension of the hierarchy downward relates to the incorporation of novel concepts. It takes place when an instance reaches a cluster and the algorithm decides the creation of a new concept to summarize the cluster instances. In this case, instances are examined to extract the concept description and then they are re-clustered below the new concept so that sub-categories can be identified in further steps. By means of this process, the construction of the conceptual hierarchy begins from scratch.

Instances sufficiently dissimilar to all clusters at terminal or non-terminal nodes in the hierarchy, create disjunct, singleton clusters. Thus, a cluster is created each time an instance cannot be included into either categories or existent clusters at a given hierarchical level. Further inclusion of instances into singleton clusters gives place to distinctive categories and, potentially, novel concepts to extend the hierarchy downward.

Both modifications, extension downward and creation of disjunct categories, take place when instances reach clusters in the hierarchy. After comparing an instance against the current cluster centroids, either the instance is placed in a new cluster given origin to a disjunct category or it is added to...
the most similar cluster among the existing ones. In the last case, the cluster is evaluated to determine whether a new concept can be defined in order to refine the hierarchy.

Merging and splitting of categories is designed to reduce the effect of instance ordering during learning. The merge operation takes two categories and combines them into a single one. It involves the creation of a new concept adding the common features of the categories being merged. In contrast to merging, splitting takes place when a concept is no longer useful to describe instances in a category. As result of splitting, a concept is removed and its children are promoted to become direct children of the parent categories. In addition to merging and splitting, the promotion of concepts to become siblings of their parent concepts is considered in order to locate concepts at the appropriate hierarchical level.

2.3.1. Concept Formation and Adaptation

WebDCC goal is to partition a set of instances into a collection of crescent specificity categories, therefore, concepts summarizing instances are defined each time the existence of multiple categories is presumed inside clusters. Thus, a lack of cluster cohesiveness is considered as a good indicator for the fact that, even though instances belong to the same cluster because of their common features, they concern to distinctive aspects of a more general category. Hence, every time a cluster updating takes place caused by the insertion of a new instance, cluster cohesiveness is evaluated to determine whether a new concept can be defined in the hierarchy to summarize this group of instances.

Because of the incremental nature of the algorithm, concept formation is defined over the instances that are part of a cluster without taking into consideration those assigned to other clusters in the hierarchy. By evaluating internal cluster properties exclusively, the algorithm does not unnecessarily retard the formation of concepts until gathering evidence about several categories. In the algorithm intended destination, which is modeling user interests, this is particularly advantageous since user interests do not develop together (e.g., if the user reads more about sports than about politics, the first concept should be created before and independently of the second one).

Intra-class similarity or cohesiveness is defined in terms of how well individual instances match the prototypical description given by the centroid of the cluster they are assigned to. It is assumed that the ability to classify instances and make inductive inferences increases with the similarity of the instances to the prototypes. A method to compute the cohesiveness of a cluster $s_r$ is to use the average pairwise similarity of instances in this cluster, i.e.

$$
\frac{1}{|s_r|^2} \sum_{d_i, d_j \in s_r} \text{sim}(d_i, d_j) = \|p_{s_r}\|^2
$$

(5)

If all document vectors in a cluster are identical, then the cohesiveness of the cluster will have the highest possible value of 1; otherwise, if the instances in a cluster vary widely, then the cohesiveness will be small, close to 0.

Mathematically, the average pairwise similarity between all instances in the cluster, including self-similarity, is equivalent to the length of the centroid vector iff all documents are scaled to be unit length. This equivalence allows the clustering algorithm to calculate the cohesiveness of clusters without revising all their instances again, but simply measuring the length of their centroid vectors.

If the cohesiveness value drops below a certain cohesiveness threshold $\varphi$, a new concept is created; otherwise, no updating in the hierarchy takes place besides the new instance incorporation. The creation of a new concept, therefore, causes the partition of instances belonging to the single category the concept represents into a set of more semantically or conceptually coherent clusters or sub-categories.

The threshold $\varphi$ impacts directly on the hierarchy shape. A high value of $\varphi$ will lead to highly branched and depth hierarchies in detriment of meaningful ones, whereas a small value of $\varphi$ will lead to more informative hierarchies including only a few meaningful concepts. In order to empirically determine a suitable value for this threshold, quantitative evaluations of hierarchy shapes were calculated using both depth and average branching factor [30] (ABF). ABF is defined as the total number of branches of all non-leaf nodes divided by the number of non-leaf nodes. Also qualitative evaluations of hierarchy shapes were obtained by measuring the accuracy of the algorithm to describe concepts in terms of the percentage of meaningful concepts out of the total number of concepts in the resulting hierarchies.

Table 1 summarizes the changes in the shape of hierarchies as the cohesiveness threshold is
varied from small values to higher ones in the Syskill/Webert and WebType collections. Each value in the table consist in the average and standard deviation of ten algorithm runs with different instance orders. In both cases, the depth and average branching of hierarchies increase along with the value of \( \varphi \). In the Syskill/Webert collection, ABF and depth are close to their optimal values, those of the target hierarchy, when \( \varphi = 0.25 \). Likewise, the best results for the WebType collection were obtained with this threshold value. For \( \varphi = 0.3 \), the number of categories and the depth of hierarchies is substantially larger than the number of categories and depth of the target hierarchies.

WebDCC accuracy in extracting concepts, on the other hand, is not the optimal for \( \varphi \) greater than 0.2; i.e., the algorithm extracts most of the target concepts and also some non-meaningful concepts. In this experiment, we considered as meaningful concepts those having a comprehensible interpretation and whose terms are related to the concept they are describing. In general terms, the higher the value of \( \varphi \), the greater the uncertainty of making inferences based on clusters and the more likely the extraction of non-meaningful concepts. In further experiments we used \( \varphi = 0.25 \) since this value generates better partitions of instances. However, to be applied in user profiling, the accuracy in the extraction of concepts is preferred to its coverage since the view users have of concepts has an strong influence in their perception of the aptitude of agents and, therefore, the trust that is granted to them.

In addition to acting as classifiers, concepts serve a descriptive purpose. For a category, its prototype or centroid vector provides a means to summarize the content of instances. Moreover, there are a relatively few terms in the centroid that account for a large fraction of its length, these terms can be quite effective in providing a description of the topics discussed within instances and, at the same time, term weights provide an indication of how central individual terms are for these topics [14]. The descriptive nature of centroid vectors has been successfully used in the past to build accurate summaries of clusters [16,27], and to improve the performance of clustering algorithms [1].

WebDCC takes advantage of the descriptive property of centroids to obtain concepts based on instances in non-cohesive clusters. In order to identify the more relevant features to describe categories, a feature selection method is applied over instances in the cluster to be summarized. This method inputs a set of features and outputs a subset of these features which is relevant to the target concept.

Two main feature selection models are commonly used in machine learning [13]: filter and wrapper models. Briefly, filter approaches are independent of the learning algorithm, whereas wrapper approaches utilize the learning algorithm as part of the evaluation scheme. The last approach should generally be preferred to the first one because the usefulness of features crucially depends on the bias of the learning algorithm. However, it is computationally expensive to apply the learning algorithm once, or even more times, for each subset of features being considered. In text classification, this disadvantage is particularly serious because of the large number of features.

WebDCC algorithm uses a filter model in which most feature selection techniques in text domains are based on [21]. An score is individually computed for each feature and, then, features are ordered according to their assigned scores. This score is given by the average frequency of terms in the cluster instances, which is available in the centroid vectors. This measure not only showed to work well in hierarchical text classification domains [23], but also can be extracted directly from the centroids without the need of reprocessing the cluster instances or prior knowledge of data about either the negative class or the entire collection of examples. Then, a feature selection threshold \( \sigma \) is defined in the \([0,1]\) range such that the weight required for a feature to be selected need to be higher than \( \sigma \) multiplied the maximum weight in the centroid. Using the maximum weight in the centroid as baseline for selection, this process becomes independent of centroid lengths, which ensures that only the more salient features are extracted.

Unlike other hierarchical representations such as decision trees, the concept hierarchies WebDCC generates are composed of polythetic classifiers, they divide instances based on their values along multiple features. Feature selection is intended to make the inductive learning more robust in facing irrelevant features. The features composing prototypes are filtered to retain those with the highest significance weights to train linear classifiers associated to the new concepts according to the aforementioned instantiation of Rocchio. In this instance-
tation, each prototype $p_c_i$ is the plain average of all training documents belonging to the category $c_i$ without taking into account negative examples. Since concepts are extracted starting from clusters, this prototype corresponds to the cluster centroid in Equation 2 preceded by local feature selection. Therefore, the linear classifier for a concept $c_{new}$ is given by the set of features or terms:

$$f(c_{new}) = \{(t_1, w_1), \ldots, (t_m, w_m)\}$$

which ranked high in the centroid of the cluster being summarized. Finally, the concept $c_{new}$ is attached with the classification threshold $\tau$ indicating the distance to which documents are considered as belonging to this category. The classification threshold $\tau$ makes a boundary so that novel instances are considered to be relevant to the category $c_{new}$ only if their similarity with $features(c_{new})$ exceeds this threshold.

Each node in the hierarchy is equipped with a classifier composed by a vector of weighted terms and a threshold. Then, starting from the highest level of the hierarchy the algorithm compares instances with nodes at each level. If the vectors representing the instance and a concept at a given level are close enough based on the cosine similarity, the instance is treated as belonging to the category the node represents and the classification continues with its child nodes. If there is not a close match, it is concluded that the instance does not belong to any category in the given hierarchical level and the classification stops.

Naturally, the value of the feature selection and classification thresholds play an important role in the hierarchical classification process. The choice of the value for the former threshold affects categorization accuracy. Even though most features might be relevant in clustering, the underlying concepts can be concisely captured using only a few features, while keeping all of them leads to a substantially detriment on accuracy. The value of the latter threshold, also impacts directly on categorization accuracy. A small value of this threshold would cause instances to be classified into wrong categories, whereas a high value would prevent instances from being classified into the correct categories.

In order to determine the effects of both thresholds in clustering solutions, we analyzed the variation in entropy for different setting of the selection and classification thresholds. The accuracy of categorization directly impacts on the entropy of the global clustering solution. If good categorization decisions are made, instances belonging to the same category will be placed in a single cluster, decreasing the entropy of the complete clustering solution. In contrast, if bad categorization decisions are made, instances will be placed in multiple clusters, increasing the entropy of solutions.

Before discussing the effect of both thresholds, the method to calculate entropy in WebDCG algorithm needs to be further explained since it is an adaptation of traditional entropy to better evaluate and compared the proposed algorithm. The entropy of an entire clustering solution is defined to be the sum of the individual cluster entropies weighted according to the size of clusters, considering all clusters in leaf nodes of the tree as in a flat clustering solution. However, WebDCG hierarchies can be evaluated at a different level of ab-

<table>
<thead>
<tr>
<th>$\varphi$</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sygalt</strong> Webert collection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABF</td>
<td>-</td>
<td>1.00 ± 0.52</td>
<td>3.70 ± 0.95</td>
<td>4.25 ± 1.16</td>
</tr>
<tr>
<td>depth</td>
<td>-</td>
<td>1.00 ± 0.00</td>
<td>1.10 ± 0.32</td>
<td>1.20 ± 0.48</td>
</tr>
<tr>
<td># concepts</td>
<td>-</td>
<td>1.00 ± 0.52</td>
<td>3.00 ± 0.74</td>
<td>5.20 ± 0.63</td>
</tr>
<tr>
<td>% accuracy</td>
<td>-</td>
<td>100 ± 0.00</td>
<td>91.00 ± 16.63</td>
<td>78.00 ± 9.96</td>
</tr>
<tr>
<td><strong>Web Type collection</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABF</td>
<td>-</td>
<td>1.50 ± 0.48</td>
<td>1.83 ± 0.39</td>
<td>2.21 ± 0.20</td>
</tr>
<tr>
<td>depth</td>
<td>-</td>
<td>2.17 ± 1.01</td>
<td>3.10 ± 0.88</td>
<td>5.80 ± 1.14</td>
</tr>
<tr>
<td># concepts</td>
<td>-</td>
<td>5.00 ± 1.49</td>
<td>10.00 ± 2.69</td>
<td>21.70 ± 3.43</td>
</tr>
<tr>
<td>% accuracy</td>
<td>-</td>
<td>97.64 ± 4.90</td>
<td>76.01 ± 6.95</td>
<td>64.74 ± 3.76</td>
</tr>
</tbody>
</table>

Table 1: Effects of the cohesiveness threshold
traction considering the partitions in the hierarchies instead of just the flat clusters. Then, entropy in WebDCC is computed by taking into account all clusters in the hierarchy leaves, but considering those clusters at terminal concepts as forming a single cluster. For example, in the hierarchy of Figure 1 instead of measuring entropy using the six leaf clusters in the global clustering solution, four clusters are used for calculation: all instances below c\textsubscript{politics} are joined into a single cluster, the same is done for c\textsubscript{football} and c\textsubscript{basketball} categories and the fourth cluster is the one placed below the root (if more clusters exist below the root or other non-terminal nodes they are also counted separately).

Figures 5 (a) and (b) show the effect of feature selection in the quality of clustering solutions for different values of the classification threshold in the Syskill\textsuperscript{R}/Webert and Web\textsuperscript{Type} collections. Irrelevant features have a negative impact in classification and, in turn, in the resulting entropy of the clustering solution. For $\sigma = 0$, the special case in which no feature selection is made, entropy reaches high values since all features are used for classification. The subsequent improvements in clustering quality that take place in the interval $0 \leq \sigma \leq 0.5$ are achieved by removing a significant number of features. Increasing $\sigma$ further has little effect on entropy as the remaining features are removed slowly. The maximum entropy is reached for $\sigma = 1$, in which case no feature is selected, so that no category is created and instances are grouped into a single level of clusters.

Figure 6(a) and (b) shows the quality of clustering solutions as the classification threshold is varied in the same collections using different values of the selection threshold. In both cases, there is a trend of entropy to reduce as the classification threshold grows, because the probability of misclassification is reduced. In other words, a small value of the classification threshold causes the precision of classification to be degraded, causing many instances to be classified into wrong categories. Thus, the resulting clusters contain instances belonging to multiple categories and their entropy is high.

However, this trend of entropy to reduce starts to be reverted as the classification threshold continues growing. If the classification threshold is too high, instances need to be extremely similar to the prototypes representing categories in order to be considered as members of these categories. Then, many instances are placed outside the categories they belong to. In terms of classification, this causes a strong decrease in recall of categorization decisions which in turn leads to an increase in entropy as it prevents clusters from being formed by all instances belonging to each category.

From the variation of both thresholds the following conclusions can be drawn. The variation in entropy caused by values of the selection threshold in the interval $0.4 \leq \sigma \leq 0.7$ is rather small. The value $\sigma = 0.5$ is the one obtaining better entropy performance not only in Figures 5 (a) and (b), but also in Figures 6 (a) and (b). However, the number of features which are selected using this value
is insufficient to precisely represent instances. The classification threshold, on the other hand, has a stronger effect in entropy, but its minimum values are clearly located in $\tau = 0.6$ in one collection and $\tau = 0.7$ in the other collection. In the following experiments, we will consider the values $\sigma = 0.4$ and $\tau = 0.65$ as default values for these parameters, but other values can be chosen taking into account the studied effects of both thresholds and their optimal intervals.

### 2.3.2. Concept Merging, Splitting and Promotion

A major issue in incremental hierarchical clustering is the sensitivity to input ordering [9]. Usually, ordering effects are mitigated by applying restructuring operators over the hierarchies, which can be broadly categorized into local and global approaches. Local approaches suffer from the inability to deal with major structural changes, but they are efficient to recover nodes misplaced at neighboring nodes. In contrast, global approaches can deal with these problems but are very expensive and make the algorithm non-incremental.

WebDCC applies three local operators that can be efficiently tested and produce better structured hierarchies for clustering and classification: merging, splitting, and promotion operators. Every time a novel concept is defined in the hierarchy, its structure is revised in order to take into account the recently acquired knowledge. Merging, splitting and promotion of concepts are considered in order to arrange the hierarchical structure and, if applied, the definition of the existent concepts is updated accordingly.

In the hierarchical revision, the promotion of the most recently created concept to higher levels of the hierarchy is first tried. To decide a concept promotion to the immediately higher hierarchical level, the algorithm tests whether the concept is vertically well-placed, this is if it is more similar to its parent than to its grandparent [12]. If the concept is not well-placed, it is promoted as a sibling of its parent concept and the algorithm continues testing this condition upward until no promotion is possible or the root is reached. Once the concept has been placed at the appropriate level of the hierarchy, merging the concept with those concepts at the same level is considered.

Merge and promotion operations are performed according to a criteria of similarity between concepts which is assessed through the overlapping descriptions of these concepts. In other words, the more concepts overlap their descriptions the more similar they are. The overlapping between two concepts $c_i$ and $c_j$ is given by:

$$\text{overlap}(c_i, c_j) = |f(c_i) \cap f(c_j)|$$ (6)

where $f(c_i)$ are the weighted terms describing $c_i$ and $f(c_j)$ are the weighted terms describing $c_j$.

Merging operation has the purpose of creating concepts to summarize instances in two or more categories. Based on the previous criteria, merging takes place when the novel concept incorporated to the hierarchy overlaps its description with one of more concepts in the same hierarchical level. Then, the concept is merged with the most similar
concept existing at this level, i.e. the concept with the greatest overlapping.

The pair of concepts \( c_i \) and \( c_j \) is merged as follows. If \( \text{overlap}(c_i, c_j) = \emptyset \), both concepts are considered well separated and no merging is performed. In contrast, if there is an intersection between concepts, i.e. \( \text{overlap}(c_i, c_j) \neq \emptyset \), the concepts are close enough to be better captured by a single generalization. A common concept parent \( c_p \) is then defined such that the features to describe \( c_p \) are those in the intersection of both categories:

\[
fe(c_p) = fe(c_i) \cap fe(c_j)
\]

(7)

Hence, \( c_p \) becomes the parent of \( c_i \) and \( c_j \), instances belonging to \( c_p \) are \( D_p = D_i \cup D_j \), the centroid \( p_c \) is given by

\[
p_c = \frac{1}{|D_p|} \sum_{d \in D_p} d
\]

(8)

and the definitions of \( c_i \) and \( c_j \) are updated to remove the terms belonging to \( c_p \), i.e. those in the intersection of both categories.

Merging two categories may cause one or both of the child categories to become useless in describing the containing instances. If the set of features describing a child category is empty after removing the features belonging to its new parent category, this child is split. This is, it is removed from the hierarchy and its children, both clusters and categories, are promoted to become direct children of the parent category.

Like merging, promotion is based on the overlapping of concepts to draw conclusions about their resemblance. If a concept is more similar to its grandparent than to its parent, it is promoted as a sibling of its parent, i.e. the concept \( c_i \) is promoted as a child of its grandparent \( c_p \) if:

\[
\frac{\text{overlap}(c_i, c_p)}{|fe(c_p)|} \geq \frac{\text{overlap}(c_i, c_j)}{|fe(c_j)|}
\]

(9)

where \( c_j \) is the parent of \( c_i \) and \( c_j \) is a child of \( c_p \). This promotion is performed if there are some terms in the parent category that are not contained in the child category. In order to promote instances to the first level of hierarchies, concepts are defined as fully overlapping with the root category. Then, \( c_p \) becomes the parent of \( c_j \), instances belonging to \( c_j \) are updated to be \( D_j - D_i \) and the corresponding prototypes are updated.

3. Experimental Results

This section details experiments we performed to evaluate the WebDCC clustering algorithm. Initial experiments with this algorithm aimed at investigating clustering solutions regarding both description and prediction tasks. Section 3.1 summarizes these experiments. Experimental results of comparing WebDCC with other clustering algorithms are presented in Section 3.2.

3.1. Evaluation of Clustering Results

WebDCC aims at finding meaningful categories or clusters to summarize instances. Like other inductive tasks, clustering solutions serve two different purposes, namely, prediction and description. In the context of user profiling, description focus on finding comprehensible user profiles, whereas prediction on guiding agent actions.

For prediction, the algorithm should be able to find the most relevant instances in the profile in order to evaluate individual recommendations. To accomplish this goal, the category the item to be recommended belongs to has to be established in the first place. Then, the recommendation algorithm evaluates the confidence in recommending the item based on the similarity of the item to the past experiences (instances) belonging to this category and the relevance of these experiences.

In order to evaluate whether the algorithm is able to identify the category an instance belongs to, we measured its accuracy in determining the label of Web pages in text collections. To compute predictive accuracy as is done for supervised classifiers, document collections with a known class structure were used to build a model holding back labels during clustering. After training, the class of test instances was predicted by classifying instances into the conceptual hierarchy and selecting the class of the most similar instance in the category the test instance was classified into. Even though accuracy at labeling instances does not measure the effectiveness in recommendation, it is useful to gain some insight about both the accuracy of hierarchical classification and the algorithm competence to retrieve the most relevant experiences to evaluate recommendations.

In all the experiments, the text collections were divided into a 70% of instances for training and a 30% of instances for testing. For the
Syskill\#Weber	collection, the split resulted in 232 training instances and 100 test instances distributed in the four categories existing in the collection. The WebType collection was divided into 649 training instances and 277 test instances and the PDDP K-series collection was divided into 1638 pages for training and 702 for testing. In order to consider ordering effects, all the results are the average of ten independent runs using random splits of each collection.

The goal of text categorization is the classification of documents into a number of predefined categories. In the hierarchical classification approach the algorithm decides the classification of instances into the most similar category at each level, if exists at least one category exceeding the classification threshold. However, each document may belong to exactly one, no category at all, or multiple categories. The fact that instances may belong to multiple categories is taken into account at prediction time, the likelihood of finding the most relevant experiences for recommendation is increased.

Figure 7 shows the results of label prediction for the mentioned collections considering both a single as well as multiple categories during classification. In the last case, each instance follows all possible paths from the root while it exceeds the classification threshold for one or more categories, until it can not be further classified down. For each followed path, an instance is retrieved and the most similar instance in the set of all retrieved instances is used to obtain the class of the target instance.

It is clear from the learning curves in the figures that the algorithm demonstrates a steady improvement in its ability to predict class labels, even though during the learning process new concepts are created and instances have to be classified in the resulting hierarchies. In the WebType collection, a reduction in predictive accuracy is observed after the creation of the first concepts, but it starts to improve as more instances are added. Finally, it is possible to conclude that the classification of instances into multiple categories increases the predictive accuracy as it was expected. However, the improvement in accuracy is small compared with the computational cost of following multiple paths in the hierarchy at prediction time.

3.2. Comparison with other Clustering Algorithms

In order to evaluate WebDCC, we compared its performance in clustering Web pages with the per-

Fig. 7. Evaluation of WebDCC predictive accuracy
formance of other clustering algorithms, including agglomerative and divisive hierarchical clustering approaches and a conceptual clustering algorithm such as COBWEB.

3.2.1. Agglomerative and Divisive Clustering

Hierarchical clustering techniques build tree structures by using either a bottom-up, agglomerative method or a top-down, divisive method. The former method begins with each instance in a distinct cluster and successively merges the two most similar clusters at each step to create a new internal node in the tree. This process continues until a single, all-inclusive root node is reached. The latter method, starts with a single cluster containing all instances and successively splits resulting clusters until only clusters of individual instances are left.

Hierarchical agglomerative clustering (HAC) [10] depends on the definition of a similarity measure between clusters to determine the pairs of clusters to be merged at each step. Numerous approaches have been developed for computing this similarity being the most common alternatives the single-link, complete-link and group-average methods. Hierarchical divisive clustering, on the other hand, is based on a global criterion function whose optimization drives the entire clustering process. These criterion functions are usually classified into four different groups: internal, external, hybrid and graph-based functions.

In order to experimentally contrast the performance of obtaining hierarchical solutions of agglomerative and divisive algorithms against WebDCC algorithm, we considered Syskille/Webert, WebType and PDDP K-series collections. The same pre-processing of instances as the one performed by WebDCC was applied to Web pages in these collections in order to generate frequency vectors with Euclidean normalization. For agglomerative algorithm, three variations were included in the comparison corresponding to the single-link (slink), complete-link (clink) and group-average (upgma) cluster similarity measures; while seven variations were included for divisive algorithms corresponding to two internal ($I_1$ and $I_2$), one external ($E_1$), two hybrid ($H_1$ and $H_2$) and two graph-based ($G_1$ and $G'_1$) criterion functions. Each of these functions and the various resulting algorithms are available in the CLUTO clustering toolkit.

The agglomerative and divisive algorithms lead to hierarchical structures called dendrograms that start with only one cluster covering all the data at its root and iteratively splits a dataset into smaller subsets until each subset consists of only one instance. The partition of these dendrograms at different levels of granularity, produce different clustering solutions. In order to make results comparable, several solutions were obtained by stopping the agglomeration process when $k$ clusters were left.

Experimental results achieved with the mentioned collections for different combination of criterion functions and values of $k$ are shown in Tables 2, 3 and 4, which summarize the entropy scores achieved in the experiments. To evaluate the various criterion functions, the performance of each function over different amount of clusters is shown, whereas the underlined entries in the table correspond to the best performing criterion function. In the tables, the performance of WebDCC is summarized by the minimum, maximum and average entropy achieved in ten algorithm runs for each collection. Also the average number of clusters and concepts extracted in the ten runs is shown.

The average accuracy using the Syskille/Webert collection is better than the one of most agglomerative and divisive solutions for approximately the same number of clusters, whereas the minimum entropy is smaller than any result obtained with the hierarchical approaches. The hierarchy corresponding to the minimum entropy value is depicted in Figure 8(a). For the WebType and PDDP K-series collections, the entropy values were inferior to those of divisive and agglomerative clustering solutions. In these collections, the method for calculating entropy caused that internal concepts (no terminal ones) increase the count of clusters because several instances are left as sibling of these concepts. In spite of this fact, WebDCC outperforms in all cases the single-link approach and its average results are in the range delimited by the minimum and maximum entropy values of the several criterion functions.

These experiments show that WebDCC is able to obtain clustering results that are comparable to the results of more computational expensive, non-incremental hierarchical algorithms. This characteristic of being non-incremental and the quadratic time complexity of agglomerative clustering, make these hierarchical approaches not applicable to the

5http://www.cs.umn.edu/~leyripis/cluto
### Table 2

Comparison of WebDCC with hierarchical approaches over the Syskill®/Weber collection

<table>
<thead>
<tr>
<th>k</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$E_1$</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$\varphi_1$</th>
<th>$\varphi_1'$</th>
<th>slink</th>
<th>clink</th>
<th>$\text{sguma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.220</td>
<td>0.210</td>
<td>0.236</td>
<td>0.271</td>
<td>0.222</td>
<td>0.235</td>
<td>0.268</td>
<td>0.946</td>
<td>0.660</td>
<td>0.046</td>
</tr>
<tr>
<td>50</td>
<td>0.108</td>
<td>0.077</td>
<td>0.115</td>
<td>0.276</td>
<td>0.105</td>
<td>0.098</td>
<td>0.104</td>
<td>0.860</td>
<td>0.398</td>
<td>0.085</td>
</tr>
<tr>
<td>60</td>
<td>0.098</td>
<td>0.069</td>
<td>0.112</td>
<td>0.276</td>
<td>0.098</td>
<td>0.095</td>
<td>0.091</td>
<td>0.763</td>
<td>0.376</td>
<td>0.068</td>
</tr>
<tr>
<td>65</td>
<td>0.090</td>
<td>0.068</td>
<td>0.109</td>
<td>0.272</td>
<td>0.096</td>
<td>0.095</td>
<td>0.091</td>
<td>0.751</td>
<td>0.071</td>
<td>0.064</td>
</tr>
<tr>
<td>70</td>
<td>0.088</td>
<td>0.067</td>
<td>0.108</td>
<td>0.272</td>
<td>0.093</td>
<td>0.091</td>
<td>0.091</td>
<td>0.732</td>
<td>0.070</td>
<td>0.057</td>
</tr>
</tbody>
</table>

WebDCC Algorithm

| # clusters | 65.4 ±7.3 | # concepts | 4.1 ±1.3 | Entropy | 0.075 ±0.025 | Min/Max | 0.027/0.106 |

### Table 3

Comparison of WebDCC with hierarchical approaches over the WebType collection

<table>
<thead>
<tr>
<th>k</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$E_1$</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$\varphi_1$</th>
<th>$\varphi_1'$</th>
<th>slink</th>
<th>clink</th>
<th>$\text{sguma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.396</td>
<td>0.354</td>
<td>0.247</td>
<td>0.327</td>
<td>0.336</td>
<td>0.328</td>
<td>0.325</td>
<td>0.882</td>
<td>0.746</td>
<td>0.879</td>
</tr>
<tr>
<td>20</td>
<td>0.302</td>
<td>0.215</td>
<td>0.204</td>
<td>0.288</td>
<td>0.210</td>
<td>0.205</td>
<td>0.292</td>
<td>0.867</td>
<td>0.456</td>
<td>0.493</td>
</tr>
<tr>
<td>100</td>
<td>0.125</td>
<td>0.125</td>
<td>0.120</td>
<td>0.120</td>
<td>0.122</td>
<td>0.122</td>
<td>0.114</td>
<td>0.761</td>
<td>0.188</td>
<td>0.201</td>
</tr>
<tr>
<td>150</td>
<td>0.100</td>
<td>0.108</td>
<td>0.099</td>
<td>0.112</td>
<td>0.096</td>
<td>0.100</td>
<td>0.097</td>
<td>0.681</td>
<td>0.105</td>
<td>0.165</td>
</tr>
<tr>
<td>170</td>
<td>0.089</td>
<td>0.103</td>
<td>0.094</td>
<td>0.102</td>
<td>0.094</td>
<td>0.103</td>
<td>0.093</td>
<td>0.660</td>
<td>0.156</td>
<td>0.157</td>
</tr>
</tbody>
</table>

WebDCC Algorithm

| # clusters | 169.5 ±27.4 | # concepts | 10.4 ±3.2 | Entropy | 0.275 ±0.047 | Min/Max | 0.193/0.257 |

### Table 4

Comparison of WebDCC with hierarchical approaches over the K-series collection

<table>
<thead>
<tr>
<th>k</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$E_1$</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$\varphi_1$</th>
<th>$\varphi_1'$</th>
<th>slink</th>
<th>clink</th>
<th>$\text{sguma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.388</td>
<td>0.368</td>
<td>0.360</td>
<td>0.359</td>
<td>0.367</td>
<td>0.378</td>
<td>0.378</td>
<td>0.858</td>
<td>0.556</td>
<td>0.519</td>
</tr>
<tr>
<td>300</td>
<td>0.169</td>
<td>0.142</td>
<td>0.138</td>
<td>0.138</td>
<td>0.137</td>
<td>0.195</td>
<td>0.151</td>
<td>0.769</td>
<td>0.166</td>
<td>0.194</td>
</tr>
<tr>
<td>580</td>
<td>0.008</td>
<td>0.089</td>
<td>0.093</td>
<td>0.088</td>
<td>0.092</td>
<td>0.140</td>
<td>0.102</td>
<td>0.556</td>
<td>0.106</td>
<td>0.123</td>
</tr>
<tr>
<td>900</td>
<td>0.054</td>
<td>0.056</td>
<td>0.061</td>
<td>0.055</td>
<td>0.062</td>
<td>0.092</td>
<td>0.083</td>
<td>0.255</td>
<td>0.066</td>
<td>0.079</td>
</tr>
</tbody>
</table>

WebDCC Algorithm

| # clusters | 580.4 ±96.7 | # concepts | 12.1 ±3.8 | Entropy | 0.195 ±0.056 | Min/Max | 0.132/0.202 |

Table 4

Comparison of WebDCC with hierarchical approaches over the K-series collection.
user profiling problem. In addition, clusters are not explicit in the output of the algorithms, but they have to be determined from the resulting dendrograms.

WebDCC, on the other hand, generates readable descriptions of the hierarchical solutions. Figures 8, 9 and 10 depict the hierarchies obtained when running the algorithm in the experiments. As a consequence of the pre-processing of documents, the terms shown are not the actual words but their stems. The different orders of instances caused the algorithm to generate different hierarchies. In some cases, more concepts than the ones existing in the target hierarchies were obtained. Nonetheless, this does not mean that the additional concepts are incorrectly extracted, but they usually belong to more specific categories existing in certain categories of the collections. For example, Figures 8(a) and (b) shown two hierarchies obtained for the Syskill@Weber collection. The hierarchy in Figure 8(b) has two sub-categories inside BioMedical, one grouping Web pages related to universities and one grouping Web pages related to international journals.

Figure 9 depicts a hierarchy obtained for the WebType collection, whereas Figure 10 shows a hierarchy summarizing the pages in the PDDP K-series collection. In both figures, it can be observed that the algorithm was able to effectively isolate fine-grained concepts in the hierarchies. For example, for the economy category in the first collection, the algorithm identified two sub-categories one related to foreign taxes and the other to public sector. Likewise, in the second figure the algorithm identified concepts as specific as Oprah Winfrey and Princess Diana accident within variety. If more instances are added, the refinement of these hierarchies can still continue starting from the leaf concepts.

It can be concluded from these results that the algorithm is able to effectively identify meaningful concepts starting from Web pages. In the unsupervised learning of hierarchies the algorithm is discovering completely new knowledge, so that also some non-meaningful concepts can appear and other concepts can be misplaced in the hierarchy. For the user profiling perspective, the generation of comprehensive models facilitates the incorporation of the user in the learning process since profiles can be read and their correctness verified by users. Thus, a user can help the algorithm to recover from bad decisions, e.g., if the user indicates that a concept does not describe his interests, the concept can be removed and its instances re-clustered in the hierarchy.

3.2.2. COBWEB

COBWEB [8] is a conceptual clustering algorithm that constructs a tree from a sequence of observations in a strict incremental scheme and employs probabilistic concept descriptions to represent the learned knowledge. The clustering solutions in COBWEB are expressed in the form of a tree, with leaves representing each instance in the tree, the root node representing the entire set of examples, and branches representing all the clusters and sub-clusters within the tree.

COBWEB has two main differences with respect to WebDCC algorithm. First, the leaves in the trees that can be obtained with COBWEB are instances themselves instead of clusters, whereas internal nodes represent groups of instances. This kind of trees are difficult to read for a user as they are deeper than those from WebDCC algorithm. Second, concepts in the tree are probabilistic description of categories. In consequence, each concept represents the probability for each attribute of taking a given value. In the case of document clustering, the attributes are the different terms in the collections, while their possible values are 1 if the term appears in the document and 0 otherwise.

The shape of the hierarchies each algorithm produces is, therefore, the main aspect to be compared. For this reason, we ran COBWEB and WebDCC algorithms over the total number of pages belonging to the Syskill@Weber collection to measure different aspects of the shape of the resulting hierarchies. Table 5 summarizes results for ten runs with different instance orders of each algorithm, the same orders were presented to both algorithms. To describe the clustering process the table shows the average and standard deviation of the number of merges and splits. The shapes of hierarchies is summarized using the total number of clusters, either leaf or internal ones, the number of internal nodes, and the hierarchy depth.

Even though the hierarchical organization of pages in the Syskill@Weber collection is very simple, since only four topics and no sub-categories are involved, COBWEB generates very deep hierarchies containing a considerable number of internal nodes, each containing a high number of features. These trees can be very hard to explore and
Fig. 8. Hierarchies for the *Sybil’s Web* collection

Fig. 9. Example of hierarchy achieved from the *WebType* collection

Fig. 10. Example of hierarchy achieved from the *PDDP K-series* collection
understand by non-expert users. In contrast, WebDCC generates hierarchies more similar in depth and number of categories to the target hierarchy as can be observed in Figure 8.

In addition to the different types of hierarchies the algorithms produced, the computational cost of obtaining a tree with COBWEB was drastically higher to the counterpart with WebDCC algorithm. This is denoted by the number of merges and splits summarized in the table and also the number of calculations the algorithm requires to decide whether to include an instance in an existing cluster, to create a new cluster, to merge or to split clusters.

<table>
<thead>
<tr>
<th></th>
<th>COBWEB</th>
<th>WebDCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of merges</td>
<td>291.8 ± 42.55</td>
<td>0.40 ± 0.52</td>
</tr>
<tr>
<td>Number of splits</td>
<td>254.8 ± 37.20</td>
<td>0.10 ± 0.32</td>
</tr>
<tr>
<td>Number of clusters</td>
<td>142.0 ± 5.85</td>
<td>208.20 ± 15.20</td>
</tr>
<tr>
<td>Number of internal nodes</td>
<td>91.4 ± 3.78</td>
<td>4.10 ± 1.28</td>
</tr>
<tr>
<td>Depth</td>
<td>13.8 ± 1.3</td>
<td>1.40 ± 0.51</td>
</tr>
</tbody>
</table>

Table 5: Comparison of WebDCC with COBWEB

4. Related work

Letizia [18], Syskill@Webert [26], Personal WebWatcher [22], NewsDude [2], Amalthea [25] and WebbMate [4] are some examples of personal assistants that help users with Web-based tasks. Most of these assistants use different methods to represent user interests coming from either the machine learning or the information retrieval communities, both communities have explored the potential of established algorithms for user modeling purposes. However, current user profiling approaches using these algorithms have several shortcomings.

In the first case, agents are engaged in a supervised learning task which requires positive and negative examples of user interests in order to generate a predictive model which is not explicitly represented but hidden in the representational formalism provided by the learning algorithm. Then, profiles could be simply a decision tree, a naïve Bayes classifier or a neural network as in Syskill@Webert, a population of vectors in a genetic algorithm as in Amalthea, or a combination of them as in NewsDude which models long-term interests with a naïve Bayes classifier and short-term interests with a k-NN classifier. Even though user profiling appears to be a good candidate for straightforward application of standard machine learning techniques, it states a number of challenges to traditional algorithms in this area [30]. For instance, the computational complexity arising from the number of features and the problem of coping with dynamic interests. More importantly, although such predictive models directly support agent decisions and offer clear and unambiguous semantics of their output formats, they act as black boxes. That is, the internal structure of the user profiles is obscure to non-expert users.

In the second case, when information retrieval algorithms have been adapted to user profiling, both documents and user interests are represented as vectors of weighted terms according to the vector-space model [32]. This is the approach followed by Letizia, Personal WebWatcher and WebbMate. User interests are, therefore, represented by either a single vector embracing all interests or, more frequently, multiple vectors representing interests in several domains. The effectiveness of profiles in this approach depends on vectors degree of generalization. Fine-grained profiles can be achieved by increasing the number of vectors, whereas coarse-grained profiles can be obtained by representing interests with a small number of vectors. In the first case, profiles effectiveness can be high and also its complexity, whereas in the second case, the effectiveness of profiles is limited because several interests coexist in a single vector. In addition, no attempt is made in information retrieval approaches to generalize the information available in a vector space.

In contrast to these approaches, user profiling based on a conceptual clustering algorithm allows the assessment of more comprehensible, semantically enhanced user profiles. This opens new possibilities regarding users interaction with their pro-
files as well as collaboration with other agents at a conceptual level.

From the user profiling perspective, the shortcomings of existing clustering algorithms to build profiles are mostly related to the clustering solutions they are able to supply, which do not resemble user interests, and to the way they build such solutions, which is generally nonincremental. In this regard, hierarchical clustering techniques build tree structures by using either a bottom-up, agglomerative method or a top-down, divisive method. Merging and splitting clusters in these approaches results in a binary tree or dendrogram that contains clustering information at many different levels. The exploration of clustering solutions requires too much insight of the user as these clustering algorithms do not make any attempt to characterize clusters. Conversely, algorithms in the conceptual clustering paradigm involved create concepts that define or exemplify the properties of each group of instances. COBWEB, for example, is an incremental conceptual clustering algorithm which constructs a tree from a sequence of observations. Each discovered concept in COBWEB records the probability of each attribute and value and is updated every time an instance is added. In contrast to WebDCC, which is based on text-learning techniques, the probabilistic concepts of COBWEB limit the application of this algorithm to document clustering because the space of features and values is highly dimensional in textual domains.

5. Conclusions

WebDCC is a conceptual clustering algorithm designed to support user profiling in personal information agents. This algorithm presents an incremental approach which allows agents interacting with users over time to acquire and maintain interest categories. Thus, agents using WebDCC can deal with unpredictable subject areas that cannot be anticipated by agent developers. In addition, WebDCC enables agents not only to discover a set of interest categories, but also to focus on the aspects of these categories users are really interested in by representing categories at different levels of specificity according to the user experience in each of them.

In contrast to existing user profiling approaches, WebDCC is concerned with descriptive aspects of the clustering results, so that it can offer to the user profiling component of agents comprehensible solutions that can be easily interpreted by users interacting with agents. Hence, agents become able of helping users to, for example, explore an information space from the perspective of their personal interests [11]. Moreover, the interaction of users with their profiles might allow agents to mitigate the problem of non-meaningful or misplaced concepts that can appear in the learning process by introducing verification points where the user can help the algorithm to recover from bad decisions. Finally, because user profiles are a starting point for creating user communities based on shared interests, meaningful profiles can help users to more easily search the profiles of other users to establish common interests.

References


