Data Min Knowl Disc (2010) 21:153–185
DOI 10.1007/s10618-010-0172-z

Hierarchical document clustering using local patterns

Hassan H. Malik · John R. Kender · Dmitriy Fradkin · Fabian Moerchen

Received: 21 March 2009 / Accepted: 6 March 2010 / Published online: 3 April 2010

Abstract The global pattern mining step in existing pattern-based hierarchical clustering algorithms may result in an unpredictable number of patterns. In this paper, we propose IDHC, a pattern-based hierarchical clustering algorithm that builds a cluster hierarchy without mining for globally significant patterns. IDHC first discovers locally promising patterns by allowing each instance to "vote" for its representative size-2 patterns in a way that ensures an effective balance between local pattern frequency and pattern significance in the dataset. The cluster hierarchy (i.e., the global model) is then directly constructed using these locally promising patterns as features. Each pattern forms an initial (possibly overlapping) cluster, and the rest of the cluster hierarchy is obtained by following a unique iterative cluster refinement process. By effectively utilizing instance-to-cluster relationships, this process directly identifies clusters for each level in the hierarchy, and efficiently prunes duplicate clusters. Furthermore, IDHC produces cluster labels that are more descriptive (patterns are not artificially

Responsible editor: Johannes Fürnkranz and Arno Knobbe.

H. H. Malik (✉)
Thomson Reuters, 195 Broadway, New York, NY 10007, USA
e-mail: hassan.malik@thomsonreuters.com; logicators@yahoo.com

J. R. Kender
Columbia University, 1214 Amsterdam, MC 0401, New York, NY 10027, USA
e-mail: jrk@cs.columbia.edu

D. Fradkin · F. Moerchen
Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA

D. Fradkin
e-mail: dmitriy.fradkin@siemens.com

F. Moerchen
e-mail: fabian.moerchen@siemens.com

restricted), and adapts a soft clustering scheme that allows instances to exist in suitable nodes at various levels in the cluster hierarchy. We present results of experiments performed on 16 standard text datasets, and show that IDHC outperforms state-of-the-art hierarchical clustering algorithms in terms of average entropy and FScore measures.

Keywords Pattern based hierarchical clustering · Interestingness measures · Dimensionality reduction · Pattern selection · Global modeling using local patterns

1 Introduction and motivation

Clustering is the partitioning of a dataset into subsets (clusters), so that the data in each subset (ideally) share some common trait. The quality of clustering achieved by traditional flat clustering algorithms (e.g., *k*-means clustering) heavily relies on the desired number of clusters (i.e., the value of *k*), which must be known in advance. Unfortunately, finding the right number of clusters is a non-trivial problem and no successful methods exist to automatically determine this value for a new, previously unseen dataset. Therefore, these algorithms require the user to provide the appropriate number of clusters. This approach, however, may be problematic because users with different backgrounds and varying levels of domain expertise may provide different values for *k*. Consequently, a clustering solution obtained by one user may not satisfy the needs of other users.

Additionally, large clusters in a flat clustering solution may not provide further insights about intra-cluster relationships, e.g., a large cluster containing instances about animals may not provide additional information to distinguish land animals from marine animals. Similarly, many small clusters may not provide further information about inter-cluster relationships.

In an attempt to avoid these problems, hierarchical clustering is widely used as a practical alternative to flat clustering. Nodes in a hierarchical clustering solution are organized in a general to specific fashion, and users have the option to analyze data at various levels of abstraction by expanding and collapsing these nodes. Most importantly, hierarchical clustering algorithms do not require the number of clusters to be known in advance.

The most successful hierarchical clustering algorithms for documents include agglomerative algorithms such as UPGMA and partitioning based algorithms such as bisecting *k*-means (Zhao and Karypis 2005). These algorithms typically represent documents with feature vectors that combine local and within-dataset frequencies of words with weighting schemes such as TFIDF (Salton and Buckley 1988). Clusters are typically represented with an appropriately normalized sum of the vector of the documents assigned to the cluster. Recently, a number of pattern-based hierarchical clustering algorithms have been proposed (Beil et al. 2002; Fung et al. 2003; Malik and Kender 2006; Xiong et al. 2004; Yu et al. 2004). These algorithms come with an added advantage of automatically identifying cluster labels, i.e., a set of words that define the cluster. Many of them easily support overlapping clusters, a common use case in document analysis where a single document might cover several distinct topics. These features are not readily available in agglomerative and partitioning based

algorithms. We identified four major problems with the existing pattern-based hierarchical clustering algorithms that are discussed in Sects. 1.2–1.5.

1.1 Notations and definitions

Let D be a dataset, $I = i_1, i_2, i_3, \ldots, i_n$ be the complete set of distinct items (i.e., binary attributes) in D. An instance X is denoted as a pair $< id, Y >$ such that id is an identifier that uniquely identifies X, and $Y \in I$ represents the set of items in X. A pattern $P = p_1, p_2, p_3, \ldots, p_n$ is a subset of I. The set of data that contains P is denoted as $D_p = \{(id, Y) \in D | P \subset Y\}$. The support of a pattern P is defined as:

$$\text{Support}(P) = \frac{|D_p|}{|D|} \tag{1}$$

P is called frequent if Support$(p) \geq$ min_sup, where min_sup is the minimum support threshold. A frequent pattern F is called closed if there exists no proper superset T with Support(T) = Support(F). The local frequency $f_{i,j}$ (also referred to as term frequency in Information Retrieval) is the number of occurrences of the considered item I_i in instance D_j. The function $ds(P, D, m)$ uses measure m to determine the significance of pattern P in dataset D. Similarly, the function $ls(P, X, m)$ uses measure m to determine the local significance of pattern P in instance X.

1.2 Problem 1: sensitivity of pattern selection to thresholds

Most of the existing pattern-based hierarchical clustering algorithms (Beil et al. 2002; Fung et al. 2003; Xiong et al. 2004; Yu et al. 2004; Malik and Kender 2006) follow a similar framework. These algorithms first mine a set of globally significant patterns, and then use these patterns to build a cluster hierarchy. The set of globally significant patterns is obtained by first calculating global significance values of a set of candidate patterns, and then selecting patterns with significance values that satisfy a user defined threshold. Common measures used to calculate global significance values include support (Beil et al. 2002; Fung et al. 2003), support with closed itemset pruning (Yu et al. 2004), h-confidence (Xiong et al. 2004), and a variety of interestingness measures (Malik and Kender 2006). Each selected pattern defines a cluster and instances are assigned to clusters if they contain the pattern. Various heuristics are applied to prune clusters and reduce or avoid overlap among clusters.

Because of their inherent dependence on the user defined global significance threshold, existing pattern-based hierarchical clustering algorithms face two potential problems. First, the set of selected globally significant patterns might not cover all instances (i.e., some instances might not contain any globally significant pattern), especially on datasets with a high degree of imbalance in topic distributions. Second, the number of globally significant patterns found heavily depends on the threshold value used. On high dimensional, highly correlated datasets with many shared patterns, the number of these patterns can be thousands of times higher than the number of instances

in the dataset. As we shown in Sect. 7.8, this is not merely a matter of elegance; the excessive number of patterns can even cause global pattern-based algorithms to fail. In our previous work (Malik and Kender 2006), we replaced minimum support with an interestingness threshold, which reduced the number of globally significant patterns. Still, there was no way to set an upper bound on the number of patterns, and the final set of global patterns sometimes did not cover all instances.

1.3 Problem 2: insensitivity to local term frequencies

Instances in real-life text and web datasets may contain a feature (i.e., an item) more than once, and these locally frequent features may better represent the main topic of the instance as compared to other, locally infrequent features. As an example, we consider a recent news article on cnn.com about certain types of dinosaurs that are believed to be good swimmers. The word "dinosaurs" occurs 19 times in the article whereas the word "marine" occurs only once. Clearly, considering both of these words with equal importance can be problematic. Unfortunately, existing pattern-based hierarchical clustering algorithms do not fully utilize these local feature frequencies. Some approaches (Fung et al. 2003; Malik and Kender 2006) use these values in scoring functions to select suitable hierarchy nodes for instances, or to select node parents. However, none of the existing pattern-based hierarchical clustering algorithms utilize a local pattern significance measure in the process of mining the initial set of patterns used for clustering. Note that the idea of having a quantitative basis instead of a binary one for pattern mining in general is not new. For example, the share measure (Carter et al. 1997) captures the percentage of a numerical total that is contributed by the items in an itemset.

1.4 Problem 3: unnecessary coupling of pattern size and hierarchy depth

Many existing pattern-based clustering algorithms (Fung et al. 2003; Yu et al. 2004; Malik and Kender 2006) tightly couple the sizes of cluster labels with the node heights in the initial cluster hierarchy. In these algorithms, the first level in the cluster hierarchy contains all size-1 patterns, the second level contains all size-2 patterns, and so on. This tight coupling is merely a consequence of the way global patterns are discovered (i.e., by first discovering size-1 patterns, which are used to form size-2 candidates, etc.), and does not necessarily reflect a real-life setting. Users would surely appreciate more descriptive cluster labels (i.e., labels that reflect the cluster structure of the dataset with all appropriate patterns, regardless of their corresponding node heights). Some of these approaches (Fung et al. 2003) later merge some child nodes with their parents if certain conditions are met, which does increase the label sizes. Still, a large percentage of nodes may remain with labels that have this property.

1.5 Problem 4: artificial constraints on overlapping clustering

Instances in real datasets may contain multiple patterns in the corresponding cluster hierarchy. As a consequence pattern-based hierarchical clustering algorithms easily

support overlapping clusters as opposed to partitional and agglomerative hierarchical clustering algorithms. The algorithms commonly require the user to provide a parameter specifying the maximum number of clusters an instance can be assigned to (Malik and Kender 2006; Yu et al. 2004) and assign this number of clusters to each instance when possible. Different documents may, however, belong to a varying number of topics and the same maximum number of topics may not be appropriate for all documents. Additionally, instead of allowing instances to exist in the most suitable clusters at any level in the hierarchy, some of these approaches first force all instances to their most specific levels (i.e., called "inner termset removal" (Malik and Kender 2006; Yu et al. 2004)), and then select the top-n (with user defined n) most suitable clusters at that level. This restriction appears to be a matter of convenience (i.e., a quick way of constraining instance duplication), and may not be useful for real-life hierarchies. For example, we consider four retirement related nodes in the open directory [1]. Two of these nodes (i.e., Business→Investing→Retirement Planning, and Home→Personal Finance→Retirement) are found at level 3, one of these nodes (i.e., Business→Human Resources→ Compensation and Benefits→401 k and Retirement) is found at level 4 and the last node (i.e., Society→People→ Generations and Age Groups→Seniors→Retirement) is found at level 5. The Open Directory is currently maintained by a large group of human editors. But if one were to automate the hierarchy generation process, the "inner termset removal" step in (Fung et al. 2003; Yu et al. 2004) would assign a general retirement related instance only to the most specific node at level 5, and eliminate this instance from nodes at levels 3 and 4, leading to counter intuitive assignments.

1.6 IDHC: a more flexible instance-driven hierarchical clustering algorithm

Instance Driven Hierarchical Clustering (IDHC) (Malik and Kender 2008) proposed here is a novel pattern-based hierarchical clustering algorithm. Instead of mining globally significant patterns and then using these patterns to construct the global model (i.e., the cluster hierarchy), IDHC applies the LeGo approach (Knobbe et al. 2008) to the hierarchical document clustering problem. IDHC first discovers locally promising patterns by allowing each instance to "vote" for its representative size-2 patterns in a way that ensures an effective balance between local pattern frequency and pattern significance in the dataset, hence simultaneously discovering local patterns and selecting the pattern set used for global modeling. The selected patterns serve as features for clustering, and each pattern defines an initial coarse cluster, which possibly overlaps with other clusters. These initial clusters are refined to obtain a cluster hierarchy with an iterative instance-driven process that avoids combinatorial explosion. Our solution has several desirable properties:

- A novel two-phased dimensionality reduction scheme ensures coverage of all instances with features.
- No global threshold for pattern selection is needed.

[1] http://www.dmoz.org/.

– Both local pattern frequency and pattern significance in the dataset are used for pattern selection.
– Initial clusters are formed using size-2 patterns that were directly selected by individual instances. This eliminates the high computational costs associated with mining longer patterns and also enables computing pattern significance in the dataset using inexpensive contingency-table-based interestingness measures.
– The number of initial size-2 patterns is guaranteed to be linear in the total number of instances in the dataset.
– The pattern length is not coupled to the hierarchy level.
– Instances can be assigned to clusters at multiple levels in the hierarchy.

We present results of experiments performed on 16 standard text datasets in Sect. 7, specifically showing in Sect. 7.3 that IDHC outperforms state-of-the-art hierarchical clustering algorithms both in terms of FScore and entropy measures (Zhao and Karypis 2005). Furthermore, we show that IDHC parameters are relatively easy to set and that the same untuned parameter values achieve high clustering quality on all datasets used in our experiments.

2 Related work

The history of pattern-based clustering goes back to the early years of data mining. Han et al. (1997) proposed a pattern-based flat clustering framework that uses association rules to generate a hypergraph of patterns (i.e., with items used as vertices and rules used to form hyperedges). An efficient hypergraph partitioning algorithm is then applied to obtain pattern clusters, and instances are clustered by assigning each instance to its best pattern cluster. This framework was later used in many applications, such as topic identification (Clifton et al. 2004). In another approach, Wang and Karypis (2004) applied efficient search space pruning techniques to obtain a global summary set that contains one of the longest frequent patterns for each transaction. This set is later used to form clusters.

Based on globally frequent itemsets, Beil et al. (2002) proposed a pattern-based hierarchical clustering framework. This framework was later enhanced by Fung et al. (2003) and Yu et al. (2004), who improved various stages of the clustering process. In a different approach, Xiong et al. (2004) first mine globally significant maximum hyperclique (i.e., with high h-confidence) patterns, and then associate instances to all applicable pattern clusters. These clusters are later merged by applying hierarchical agglomerative clustering (i.e., UPGMA), which was also used by Fung et al. (2003) and Yu et al. (2004) to merge top level nodes. Results in Xiong et al. (2004) show that this approach results in clustering quality that is similar to UPGMA, with an added advantage of automatically identifying cluster labels. In Li and Chung (2005), frequent word sequences are found using Generalized Suffix Tree and are used to generate initial clusters consisting of documents supporting these sequences. The clusters are merged based on sequence similarity and cluster overlap.

In our previous work Malik and Kender (2006), we improved the framework in Beil et al. (2002), Fung et al. (2003), Yu et al. (2004) by using closed interesting itemsets as globally significant patterns used for clustering, and by using an interestingness

measure to efficiently select hierarchical relationships. We showed that this approach outperformed both existing pattern-based hierarchical clustering algorithms, and the best known agglomerative (i.e., UPGMA; Zhao and Karypis 2005) and partitioning based (i.e., bisecting k-means with I_2 criterion function; Zhao and Karypis 2005) algorithms on 9 commonly used datasets. All previous global pattern-based approaches, including ours, suffer from many of the limitations discussed in Sect. 1.

Our work also relates to subspace clustering, "an extension of traditional clustering that seeks to find clusters in different subspaces within a dataset. Subspace clustering algorithms localize the search for relevant dimensions allowing them to find clusters that exist in multiple, possibly overlapping subspaces" (Parsons et al. 2004). These algorithms either follow a top-down, or a bottom-up search strategy. Top-down algorithms find an initial clustering in the full set of dimensions and evaluate the subspaces of each cluster, iteratively improving the results. On the other hand, bottom-up algorithms find dense regions in low dimensional spaces and combine them to form clusters.

3 Dimensionality reduction

Large real life document collections often suffer from high dimensionality, and this dimensionality varies greatly. For example, the commonly used Reuters 21578 dataset,[2] which uses a restricted vocabulary, contains over 19 K unique features in less than 11 K news documents. More realistic news data from high frequency news streams may contain more than 150 K features in less than 20 K documents (Moerchen et al. 2007). Therefore, reducing the dimensionality of the feature space can significantly improve the performance of pattern-based clustering algorithms, as the number of discovered patterns directly depends on the number of initial items. The availability of a labeled training set in supervised problems like classification allows for applying more sophisticated dimensionality reduction (i.e., feature selection) techniques, such as Information Gain. In contrast, there is limited information (i.e., feature frequencies in the dataset and in the individual documents) available in unsupervised problems such as clustering.

Popular unsupervised feature selection methods that has successfully been applied to text data include Document Frequency thresholding (DF) and Term Strength (TS) (Yang and Pedersen 1997). Document Frequency thresholding is an efficient method that selects features that occur in less than a user-defined number of documents. On the other hand, Term Strength first computes pairwise similarities of all document pairs in the corpus, keeping pairs with similarities above a user-defined threshold. The Term Strength is then computed based on the conditional probability that a term occurs in the second half of a pair of related documents given that it occurs in the first half. As others have also noted (Yang and Pedersen 1997), TS is less practical for large scale problems because of its time complexity that is quadratic in the number of instances.

Since both DF and TS use global thresholds, the set of features selected by these methods is not guaranteed to cover all instances (i.e., some instances might not contain

[2] http://kdd.ics.uci.edu/databases/reuters21578.

any selected features) contributing to the problem discussed in Sect. 1.2, unless the global thresholds are very low, in which case feature selection step is not very useful. To address the need to reduce dimensionality in an efficient manner while attempting to ensure coverage, we adapt a novel two-phased heuristic approach in this paper:

Step 1 (select initial features): Apply DF thresholding to select the globally most useful features by selecting features that are neither too frequent nor too infrequent. i.e., by eliminating features that occur in more than *MaxS* or less than *MinS* documents. Experiments reported in this paper selected features that existed in less than or equal to 95% of the instances, and at least two of them in the default setting.

Step 2 (ensure local coverage): For each instance i in the dataset, first sort all features in i in the decreasing order of their local frequencies. Next, select the top-n highest frequency features and add them to the set of selected features. All experiments reported in this paper used a fixed value of $n = 10$, ensuring that each document is represented by at least 10 locally most frequent features, thereby resolving coverage and insensitivity to local term frequency problems described in Sects. 1.2 and 1.3. This value of n was selected intuitively and replacing it with any other value is expected to have very little effect on the results in our default setting. (This is empirically demonstrated in Sect. 7.6). This is because the upper bound of 95% used in the previous step is rarely met, and the lower bound of two documents leads to discarding only features that exist in a single document. Section 7.7 provides additional results that demonstrate the benefit of ensuring coverage in non-default settings.

4 Instance-driven hierarchical clustering

As discussed in Sect. 1.2, the threshold-based global pattern mining step in existing pattern-based hierarchical clustering algorithms may result in an unpredictable number of patterns, with no coverage guarantees. The IDHC algorithm (Algorithm 1) adapts a novel LeGo-based (Knobbe et al. 2008) approach to address these problems. Before proceeding, we highlight how various stages in the IDHC algorithm fits within the LeGo framework. LeGo has three main phases:

1. Local pattern discovery
2. Pattern set discovery
3. Global modeling

The local pattern discovery phase produces a (potentially large) set of candidate local patterns. The pattern set discovery phase selects a subset of these patterns that are informative and relevant and show little redundancy. Finally, the global modeling phase turns these patterns into a well-balanced global model.

The first stage in IDHC (Sect. 4.1) selects significant patterns with respect to each instance. All size-2 patterns in the instance are considered as candidates (LeGo phase 1). Local pattern frequency and pattern significance in the dataset are used to compute a score for each candidate pattern. Some of the top scoring patterns are then selected as initial features for global modeling (LeGo phase 2). The second stage in IDHC (Sect. 4.2) further refines the pattern set by pruning redundant patterns (LeGo phase 2). Finally, the third stage in IDHC (Sect. 4.3) constructs the cluster hierarchy by first forming an initial cluster for each pattern and then applying an iterative refinement process to obtain the rest of the cluster hierarchy (LeGo phase 3). Computational time required by this stage is reduced by maintaining a relationship between instances and their selected local patterns.

We now explain the three major stages in the IDHC algorithm, the pseudocode for which is given in Algorithm 1.

Algorithm 1: The IDHC algorithm

```
Input: dataset, patternSelectionScheme, minStdDev, k, measure
reduce dimensionality as explained in Sect. 3
top_level_clusters = ∅
instance_cluster_pointers = ∅
for transaction t ∈ dataset do
    list = ∅
    for size-2 pattern p ∈ t do
        s_l = ls(p, t, LOCAL_FREQUENCY)
        s_d = ds(p, dataset, measure)
        significance(p) = s_l * s_d
        append(list, p)
    end
    sort list in decreasing order of significance values
    update_clusters(top_level_clusters, instance_cluster_pointers, t, list)
end
prune-duplicates(instance_cluster_pointers, top_level_clusters)
clusters_to_refine = top_level_clusters
while size(clusters_to_refine) > 0 do
    refined_clusters = refine-clusters(instance_cluster_ptrs, clusters_to_refine)
    regenerate instance_cluster_pointers using refined_clusters
    prune-duplicates(instance_cluster_pointers, refined_clusters)
    clusters_to_refine = refined_clusters
end
apply bisecting k-means to merge top_level_clusters
```

4.1 Stage 1: select significant patterns with respect to each instance

Existing pattern-based hierarchical clustering algorithms use frequent itemsets or their variants as a basis for clustering (Sect. 1.2). Frequent itemset mining, closed frequent itemset mining and even simpler problems associated with itemset mining (such as deciding whether there is a frequent itemset of cardinality k in database D) has been proven NP-Complete or NP-Hard (Angiulli et al. 2001; Gunopulos et al. 2003; Wu 2006). Therefore, itemset mining algorithms rely on heuristics that work well in the common case, but may fail on highly correlated datasets (Sect. 7.8). Even in the common case, these heuristics may result in an excessive number of itemsets (Sect. 1.2). Often, the number of candidate itemsets considered as well as the number of final itemsets found may significantly increase with the itemset sizes.

Intuitively, a pattern-based hierarchical clustering algorithm should not need any more than $O(n)$ candidate patterns to cluster a dataset with n instances. However, as a negative side effect of mining patterns without considering the actual task in hand (i.e., clustering), existing algorithms are often forced to deal with hundreds or even thousands of times more patterns than the number of instances in the target dataset.

IDHC adapts an instance-driven approach to avoid these problems. This approach allows each instance to select its representative size-2 patterns and guarantees that the total number of mined patterns is linear in the number of instances. A size-2 pattern is essentially a pair of items (i.e., words) that co-occurs in a document. Size-2 patterns resemble bigrams in Natural Language Processing, with a difference that they do not need to be contiguous to each other. We restricted the initial pattern mining to size-2 patterns for three reasons:

- Efficiency: Size-2 patterns can be mined far more efficiently than longer patterns because the number of size-2 pattern candidates is often orders of magnitude less than the number of longer-sized pattern candidates. For a dataset with F features, instances may have up to $O(F^2)$ size-2 pattern candidates. However, this theoretical upper bound is almost never met on document datasets because of their inherent sparseness. Furthermore, real-life applications often restrict the number of features considered in an instance to a small constant $C : C \lll F$, i.e., by truncating features that appear after the Cth feature or by selecting top-C locally most frequent features. With this restriction in place, size-2 patterns for all instances can be mined in time that is linear in the number of instances.
- Computing Pattern Significance: Common statistical interestingness measures (Geng and Hamilton 2006; Tan et al. 2002) use 2×2 contingency tables to compute correlation or interdependence between two variables. This allows computing the significance of size-2 patterns in constant time (assuming pattern frequencies are readily available). Computing significance of longer patterns either requires making heuristic assumptions (Malik and Kender 2006) or using substantially more expensive measures such as log-linear analysis (Brijs et al. 2003) that uses multi-way contingency tables.
- Since the primary task is clustering, there is no reason to generate a large number of longer patterns because only a small percentage of these patterns are likely to form meaningful clusters. Once size-2 patterns are available for all instances, they

can be easily merged to form longer patterns (and their corresponding clusters) in a controlled fashion (Sects. 4.2 and 4.3).

Therefore, after reducing the dimensionality of the feature space and initializing the necessary data structures (lines 4-4 of Algorithm 1), we process instances in the dataset in a purely local way (i.e., on an instance by instance basis). Each size-2 pattern in an instance is processed (lines 4-4) to compute its "overall" significance with respect to the current instance, considering the pattern significance at both local and dataset levels.

First, we use the "ls" method in Algorithm 2 to determine the local pattern significance. We used local frequency as the local significance measure (i.e., features that occur more frequently in the document are considered more important with respect to the document). Since each size-2 pattern contains two features, we calculate local pattern significance by averaging the local frequencies of both of the features (i.e., pattern1 and pattern2) in the size-2 pattern (i.e., pattern) as it favors patterns that contain two locally frequent features. We have considered using maximum or minimum as alternatives to averaging. However, these approaches assign the same values to a large number of patterns and do not provide any further discriminative information. Our choice was also inspired by Tan et al. (2002) that used averaging to compute symmetric interestingness scores.

Next, we call the "ds" method in Algorithm 2, which uses a common interestingness measure to determine the pattern significance in the dataset. In Malik and Kender (2006), 22 interestingness measures previously proposed in Geng and Hamilton (2006), Tan et al. (2002), were evaluated in the context of global pattern-based hierarchical clustering. Only a small number of these were found to be stable across all datasets used in the evaluation. We have found that the same measures are useful to determine the pattern significance in the dataset in this context. Table 1 presents the formulas of interestingness measures considered. All these measures are functions of 2×2 contingency tables for binary variables A and B indicating presence or absence of particular features. A detailed comparison of the properties of these interestingness measures is given in Geng and Hamilton (2006), Tan et al. (2002). Algorithm 2 describes computation of "ls" and "ds".

Table 1 Interestingness measures for size-2 patterns (A, B) (see Sect. 7.4 for empirical evaluation)

Symbol	Interestingness measure	Formula
AV	Added value	$\max(P(B\|A) - P(B), P(A\|B) - P(A))$
CF	Certainty factor	$\max\left(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(B)}{1-P(A)}\right)$
V	Conviction	$\max\left(\frac{P(A)P(\neg B)}{P(A\neg B)}, \frac{P(\neg A)P(B)}{P(\neg AB)}\right)$
χ^2	Chi-squared	$\sum_{ij} \frac{P(A_i B_j)-P(A_i)P(B_j)}{P(A_i)P(B_j)}$
YQ	Yule's Q	$\frac{P(AB)P(\neg A\neg B)-P(\neg AB)P(A\neg B)}{P(AB)P(\neg A\neg B)+P(\neg AB)P(A\neg B)}$
MI	Mutual information	$\frac{\sum_{ij} P(A_i B_j)\log_2\left(\frac{P(A_i B_j)}{P(A_i)P(B_j)}\right)}{\min\left(-\sum_i P(A_i)\log_2(P(A_i)), -\sum_j P(B_j)\log_2(P(B_j))\right)}$

Algorithm 2: Functions for computing local and within-dataset significance of size-2 patterns. These methods reflect the reference implementation used in this paper, many alternative implementations are possible

```
1 function ls(pattern, instance, measure)
2 begin
3 |   return average(freq(pattern 1, t), freq(pattern 2, t))
4 end
5 function ds(pattern, dataset, measure)
6 begin
7 |   compute the contingency table using pattern 1 and pattern 2 as variables
8 |   apply measure on the contingency table and return the outcome
9 end
```

Once both values are available, we compute the overall pattern significance with respect to the current instance (line 9 in Algorithm 1) by multiplying local and within-dataset pattern significance values. Local frequencies and interestingness measures do not follow similar distributions and local frequencies in general have much larger magnitudes. Using combination schemes other than product (such as average or sum) would have resulted in local frequencies dominating the scores, or require introducing an additional normalization step which would complicate the algorithm. Since we are only concerned with the relative order of patterns and not the actual significance values, we use product as a simple way of combining local and within-dataset pettern significance values.

All size-2 patterns are then sorted in decreasing order of their overall within-instance significance values (line 12 in Algorithm 1), and a local pattern selection scheme is applied to select representative patterns with respect to the current instance (line 13). We evaluate two schemes, shown in Algorithm 4 for this purpose. The first scheme (line 6 in Algorithm 4) selects a variable number of patterns for each instance by selecting up to *maxK* patterns with significance values that are greater than or equal to "*minStdDev*" standard deviations from the mean, where *maxK* and *minStdDev* are user definable parameters. In contrast, the second scheme (line 14 in Algorithm 4) always selects the top-*k* patterns with highest scores. This scheme essentially allows the user to limit the degree of cluster overlap. For either scheme, we ensure coverage and account for boundary conditions (i.e., instances with a very small number of patterns) by also always selecting the most-significant pattern (line 3 in Algorithm 4).

Each unique size-2 pattern selected by some document forms an initial cluster, and instances are associated with the pattern clusters they selected. We maintain a list of pointers for each instance to track instance-to-cluster relationships.

4.2 Stage 2: prune duplicate clusters

The set of initial clusters may contain duplicates (i.e., clusters with different labels but the exact same instances). Since these redundant clusters are unlikely to add any

Algorithm 3: Supporting methods for the IDHC algorithm

```
1  function add-to-cluster(cluster_list, pattern, instance)
2  begin
3      if pattern ∉ cluster_list then
4          add a new cluster with label = pattern to cluster_list
5      append instance to cluster with label = pattern in cluster_list
6  end
7  function prune-duplicates(instance_cluster_ptrs, cluster_list)
8  begin
9      for ptr lists l in instance_cluster_ptrs do
10         m = clusters in l that also exist in cluster_list
11         for cluster pairs p(cluster1, cluster2) ∈ m do
12             if cluster1 and cluster2 contain same instances then
13                 label(cluster1) = merge-labels(cluster1, cluster2)
14                 remove(cluster_list, cluster2)
15 end
16 function refine-clusters(instance_cluster_ptrs, cluster_list)
17 begin
18     refined_clusters = ∅
19     for ptr lists l ∈ instance_cluster_pointers do
20         m = clusters in l that also exists in cluster_list
21         for cluster c ∈ m do
22             if size(c) < 2 then
23                 remove c from m
24         for cluster pairs p(cluster1, cluster2) ∈ m do
25             cluster_label = merge-labels(cluster1, cluster2)
26             if cluster_label not in refined_clusters then
27                 Add a new cluster to refined_clusters with label = cluster_label
28             common_instances = cluster1 ∩ cluster2
29             add instances in common_instances to cluster with label =
               cluster_label in refined_clusters
30             mark instances in common_instances for elimination in both cluster1
               and cluster2
31             Add cluster with label = cluster_label as a child node to both cluster1
               and cluster2
32     In one pass over clusters in cluster_list, prune all instances that were marked
       for elimination
33     Return refined_clusters
34 end
```

practical value to the user, we merge / prune them in a way that enhances the label of the retained unique cluster. The naïve way of performing this operation requires comparing each cluster with all other clusters (quadratic time). Fortunately, as a positive side effect of our instance-driven approach, we already know instance-to-cluster relationships. It is not difficult to prove that checking for and pruning duplicate clusters locally (i.e., by comparing cluster pairs in each instance's pointer list, and repeating this process for all instances) also prunes all global duplicates.

Method "prune-duplicates" in Algorithm 3 implements pruning based on this observation, avoiding quadratic time cluster comparisons. While removing duplicate clusters, it at the same time expands cluster labels by merging the label of the retained cluster with the duplicate cluster being pruned. This results in increasingly meaningful (more specific) labels, partially addressing the problem described in Sect. 1.4. All the non-redundant clusters remain unchanged.

Algorithm 4: This function updates current set of clusters and instance pointers with document t and its select patterns, stored in list

```
1  function update_clusters(top_level_clusters, instance_cluster_pointers, t, list)
2  begin
3      add-to-cluster(top_level_clusters, list(1), t)
4      append(instance_cluster_pointers(t) , list(1))
5      switch patternSelectionScheme do
6          case MIN_STD_DEV
7              calculate mean and standard_deviation of significance values in list
8              min_significance = mean + (standard_deviation * minStdDev)
9              maxK = k
10             for i = 2; i ≤maxK; i++ do
11                 if significance(list(i)) < min_significance then break
12                 add-to-cluster(top_level_clusters, list(i), t)
13                 append(instance_cluster_pointers(t), list(i))
14         case TOP_K
15             k = min(k, size(list))
16             for i = 2; i ≤k; i++ do
17                 add-to-cluster(top_level_clusters, list(i), t)
18                 append(instance_cluster_pointers(t), list(i))
19 end
```

4.3 Stage 3: generate the cluster hierarchy

The initial clusters form the top level nodes in the cluster hierarchy, and the rest of the hierarchy is obtained by following an iterative cluster refinement process (lines 4-4 in Algorithm 1) that makes patterns progressively longer and cluster memberships progressively sparser. Atomic clusters (i.e., clusters with only one instance) clearly can not be any more specific and do not need to be considered for refinement. In fact,

refinement is only necessary when a cluster c1 shares some instances with another cluster c2. These common instances can be removed from both c1 and c2, and added to a node that is a child to both of these nodes. This refined node still retains the instance memberships of the originating clusters for retrieval purposes (i.e., as child nodes are merely specializations, and are considered part of their parents). This determination of overlap exploits instance-to-cluster pointers in a way similar to our duplicate cluster pruning scheme in Sect. 4.2, resulting in an efficient procedure.

Method "refine-clusters" in Algorithm 3, based on the above observations, finds clusters for the next level. For this purpose, on an instance by instance basis, it first identifies cluster pairs from non-atomic clusters that share some instances. Next, it appends these shared instances to a child cluster, the label of which is obtained by merging labels of the cluster pair itself.

A child cluster with a "merged" label may already exist, for two possible reasons. First, the same cluster pair may have existed in the pointer list of another instance that has already been processed. Second, merging labels of two different cluster pairs may result in a single label. However, it is easy to see that first appending shared instances to the cluster with the resulting label, and then adding this cluster as a child to both the originating clusters does not affect instance memberships of the parent clusters. One final note on implementation: since clusters may share instances with several other clusters, deleting shared instances from originating clusters immediately after adding them to a new child cluster may result in eliminating some valid child clusters. Therefore, shared instances are marked for elimination as they are found, but pruned only after all cluster pairs are processed.

Hierarchy refinement continues from level to level. Efficiency is maintained by tracking pointers to newly generated clusters. These pointers are later used to regenerate instance-to-cluster pointers (line 19 of Algorithm 1) in one pass over the newly generated clusters. Since newly generated clusters can contain duplicates, we apply the duplicate cluster pruning process (Sect. 4.2) in each iteration (Line 4 of Algorithm 1). The full process is repeated until all clusters are refined (i.e., all leaf-clusters are either atomic or contain instances that are not shared with any other cluster at their level). This refinement process eliminates the need to mine all patterns of a specific size and only mines patterns that are likely to increase the specificity of some general clusters at the current level. As a result, the final hierarchy contains smaller but more specific clusters as we move further from the root. Note that the degree of cluster overlap in the resulting hierarchy depends on the degree of overlap in the initial size-2 patterns selected by instances.

Pattern-based clustering algorithms can result in a large number of top level clusters. Most of the existing algorithms (Beil et al. 2002; Fung et al. 2003; Yu et al. 2004) use agglomerative clustering to merge these top level nodes. In Malik and Kender (2006) agglomerative clustering was replaced with bisecting k-means (using I_2 criterion function; Zhao and Karypis 2005) for computational efficiency. Following a similar approach, IDHC first obtains a joint feature vector for each top level cluster in the hierarchy by combining the feature vectors of all documents in the cluster, and then applies bisecting k-means on these vectors. It is important to note that using bisecting k-means at this stage does not require introduction of a new parameter for desired number of clusters. Bisecting k-means repeatedly partitions documents into

two clusters and then selects one of these clusters for further bisection. On a dataset with n instances, this process continues n - 1 times, leading to n leaf clusters, each containing a single document (Sect. 3 in Zhao and Karypis (2005)). Since bisecting k-means always partitions each cluster into two clusters it does not use parameter k and produces a dendrogram that is similar to the output of hierarchical agglomerative clustering. In our case therefore, bisecting k-means will proceed until all top-level level clusters from the previous stage are separated from each other, because each top-level cluster forms a single instance for bisecting k-means.

We finally note that the merging of top-level clusters does not depend on the rest of the functionality explained in this stage. Therefore, it can be applied in parallel to or in the beginning of this stage, if desired.

5 Example

We would like to illustrate some aspects of the workings of IDHC algorithm with an example. Table 2 provides a sample transaction dataset with documents about cars and felines. Using Added Value (Geng and Hamilton 2006; Tan et al. 2002) as the interestingness measure and the standard deviation-based pattern selection scheme (with *minStdDev* = 1.0), we obtain the most significant patterns with respect to each instance as shown in Table 3. These patterns are used to form the initial clusters in Fig. 1, which also shows instance-to-cluster pointers. For demonstration purposes, this example used a small value for *minStdDev*, which results in a relatively high number of initial patterns. Experiments in Sect. 7.3 used a more realistic value for this parameter.

Tables 2 and 3 also demonstrate how the algorithm "balances" local pattern significance and pattern significance in the dataset. As an example, instance "T4" contains one item (i.e., 'roar') with local frequency = 4, three items (i.e., 'jaguar', 'feline' and 'cheetah') with frequency = 3, two items (i.e., 'power' and 'tiger') with frequency = 2, and one item (i.e., 'speed') with frequency = 1. A pattern selection scheme that only

Table 2 Instance database used as an example

ID	Features and local frequencies	# Size-2 patterns
T1	(car:2), (jaguar:4), (power:1), (quiet:2), (feline:4), (tiger:1)	15
T2	(car:3), (speed:1), (power:6), (roar:1), (engine:4)	10
T3	(jaguar:2), (speed:3), (power:1), (ride:5), (cheetah:2)	10
T4	(jaguar:3), (speed:1), (power:2), (roar:4), (feline:3), (cheetah:3), (tiger:2)	21
T5	(jaguar:7), (speed:2), (power:1), (quiet:3), (ride:2)	10
T6	(car:1), (jaguar:1), (speed:1), (roar:1), (feline:3), (cheetah:1))	15
T7	(jaguar:9), (speed:3), (drive:4), (quiet:5), (feline:1), (tiger:5)	15
T8	(speed:6), (power:2), (engine:1), (ride:1), (cheetah:3)	10
T9	(jaguar:3), (power:2), (feline:4), (cheetah:1), (tiger:8)	10
T10	(car:4), (jaguar:2), (power:7), (drive:3), (ride:6)	10
T11	(speed:1), (roar:1), (drive:1), (engine:2), (quiet:1), (ride:4), (feline:1)	21

Table 3 Selected patterns for the example

ID	Selected patterns
T1	(jaguar, feline) (feline, tiger), (quiet, feline)
T2	(power, engine), (car, power)
T3	(speed, ride)
T4	(roar, feline) (feline, tiger)
T5	(jaguar, quiet) (jaguar, power)
T6	(roar, feline) (jaguar, feline) (feline, cheetah)
T7	(jaguar, tiger)
T8	(speed, engine) (speed, cheetah)
T9	(feline, tiger) (jaguar, tiger)
T10	(power, ride) (drive, ride)
T11	(engine, ride) (drive, ride) (speed, ride) (quiet, ride)

considers local significance would rank size-2 patterns that include two of {'roar', 'jaguar', 'feline' and 'cheetah'} higher than the other size-2 patterns in this instance. Similarly, considering the values in Table 4, a pattern selection scheme that only considers pattern significance in the dataset would rank patterns ('feline', 'tiger') and ('jaguar', 'tiger') higher than the other patterns. The final set of patterns selected for this instance (i.e., ('roar', 'feline') and ('feline', 'tiger')) does include the most frequent local item (i.e., 'roar'), but does not include two of the three items with frequency = 3. Instead, the algorithm selects pattern ('feline', 'tiger') that has a higher Added Value, providing a better balance between local pattern significance and pattern significance in the dataset.

Finally, we observe that the number of patterns selected by our standard deviation based scheme is not necessarily proportional to the number of size-2 patterns available in an instance. As an example, both T4 and T11 contain 21 size-2 patterns but our scheme selected twice as many patterns for T11.

Considering the 17 initial clusters in Fig. 1, the naïve way of identifying duplicate clusters will need up to 136 cluster-pair comparisons. Using instance-to-cluster relationships reduces the number of these comparisons to no more than 18 (i.e., we perform only three cluster pair comparisons for T1; {('quick', 'feline'), ('jaguar', 'feline')}, {('quick', 'feline'), ('feline', 'tiger')} and {('jaguar', 'feline'), ('feline', 'tiger')}). After processing all instances, we easily identify four duplicates (marked in boxes in Fig. 1). These duplicates are pruned and their labels are merged to obtain the 13 clusters in the left side of Fig. 2.

The right side of Fig. 2 demonstrates refining clusters (from the left side of the same figure) to the next level. Processing cluster pointers from T1, we find only one pair of non-atomic clusters (i.e., {('jaguar', 'feline'), ('feline', 'tiger')}), with T1 itself as the only shared instance. We then merge labels of the cluster pair to obtain ('jaguar', 'feline', 'tiger'), which is used to form the new child node. Cluster pointers from T2 to T11 are processed in a similar fashion to obtain 4 clusters for the next level. This process may result in adding several children to the same cluster (i.e., two child clusters

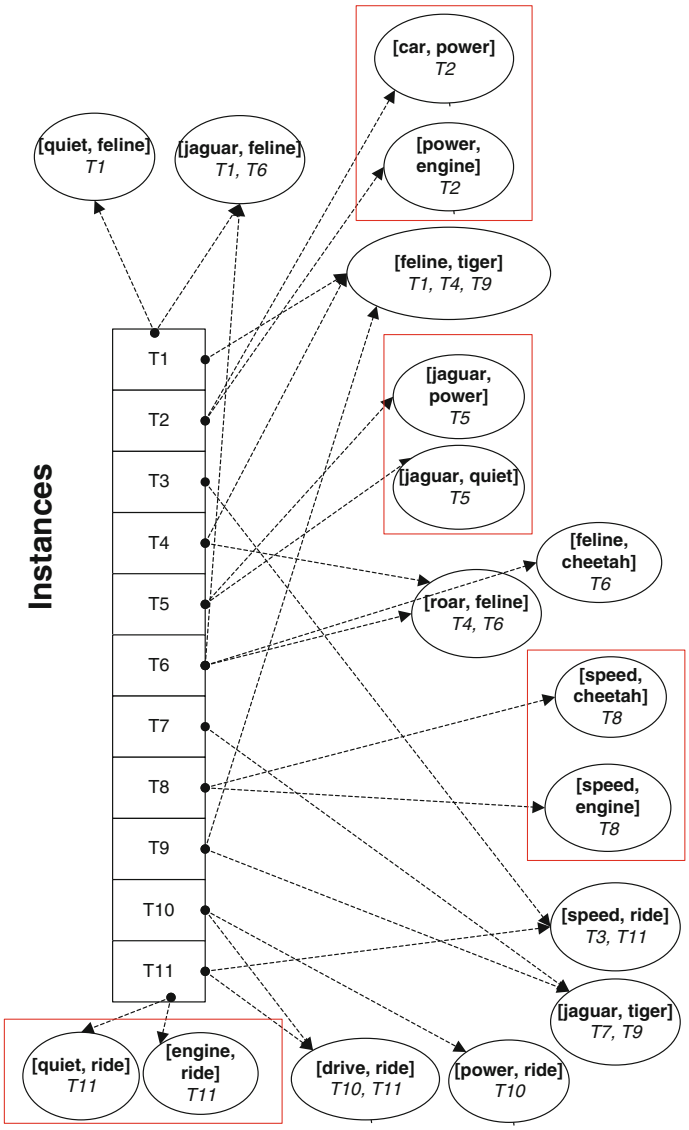

Fig. 1 Initial clusters. Candidates for deletion are boxed

added to ('roar', 'feline')) or appending several instances to an existing child cluster (i.e., two instances added to cluster ('jaguar', 'feline', 'tiger')).

6 Discussion

On the surface, it might seem like IDHC merely replaces some global thresholds (i.e., minimum support (Beil et al. 2002; Fung et al. 2003; Xiong et al. 2004; Yu et al.

Table 4 Within-dataset significance values of some size-2 patterns using Added Value (AV) transformed to positive scale

Pattern	AV
(jaguar,power)	0.52
(roar, feline)	0.70
(jaguar, roar)	0.38
(roar, cheetah)	0.54
(jaguar, feline)	0.60
(roar, tiger)	0.38
(jaguar, cheetah)	0.57
(feline, cheetah)	0.55
(jaguar, tiger)	0.77
(feline, tiger)	0.95
(power, roar)	0.38
(cheetah, tiger)	0.54

2004) or minimum interestingness (Malik and Kender 2006), and maximum instance duplication (Malik and Kender 2006; Yu et al. 2004) with a set of local thresholds (i.e., *maxK* and *minStdDev* for the standard deviation based pattern selection scheme, or *k* for the top-*k* pattern selection scheme). However, as we have discussed in Sect. 1, selecting a dataset-independent value for any of the commonly used global thresholds (i.e., minimum support) is non-trivial. Any selected value can result in a very large or a very small number of patterns, with no upper bound on the number of patterns mined. In contrast, we show in Sect. 7.3 that our local parameters are rather stable across datasets.

Additionally, we note that the computational complexity of calculating within-dataset pattern significance using an interestingness measure (line 8 of Algorithm 1) depends on the underlying dataset representation used for frequency counting. Experiments in this paper represented datasets as compressed bitmaps (Malik and Kender 2007), with frequency counting time complexity of O (d), where d is the number of documents in the dataset. However in practice, frequency counting using compressed bitmaps takes close to constant time on document datasets because of their inherent sparseness. Similarly, the sparseness of features limits the number of size-2 patterns considered for each instance even though it can theoretically be quadratic in the number of features in the entire dataset (Sect. 4.1).

Finally, we note that in practice, IDHC scales linearly with the number of instances on real-life text datasets. Informally analyzing major steps in the IDHC algorithm, the first step of selecting size-2 patterns for instances (Sect. 4.1) scales linearly because the number of size-2 patterns considered for a given instance always remains the same. Further, adding instances to the dataset can only add a linear number of bits to the compressed bitmaps used for pattern frequency counting. Next, the pruning of duplicate clusters (Sect. 4.2) scales linearly because its processing of instance-to-cluster pointers is entirely local and instance by instance. The iterative hierarchy refinement process in Sect. 4.3 also scales linearly for the same reason. Note that the number of

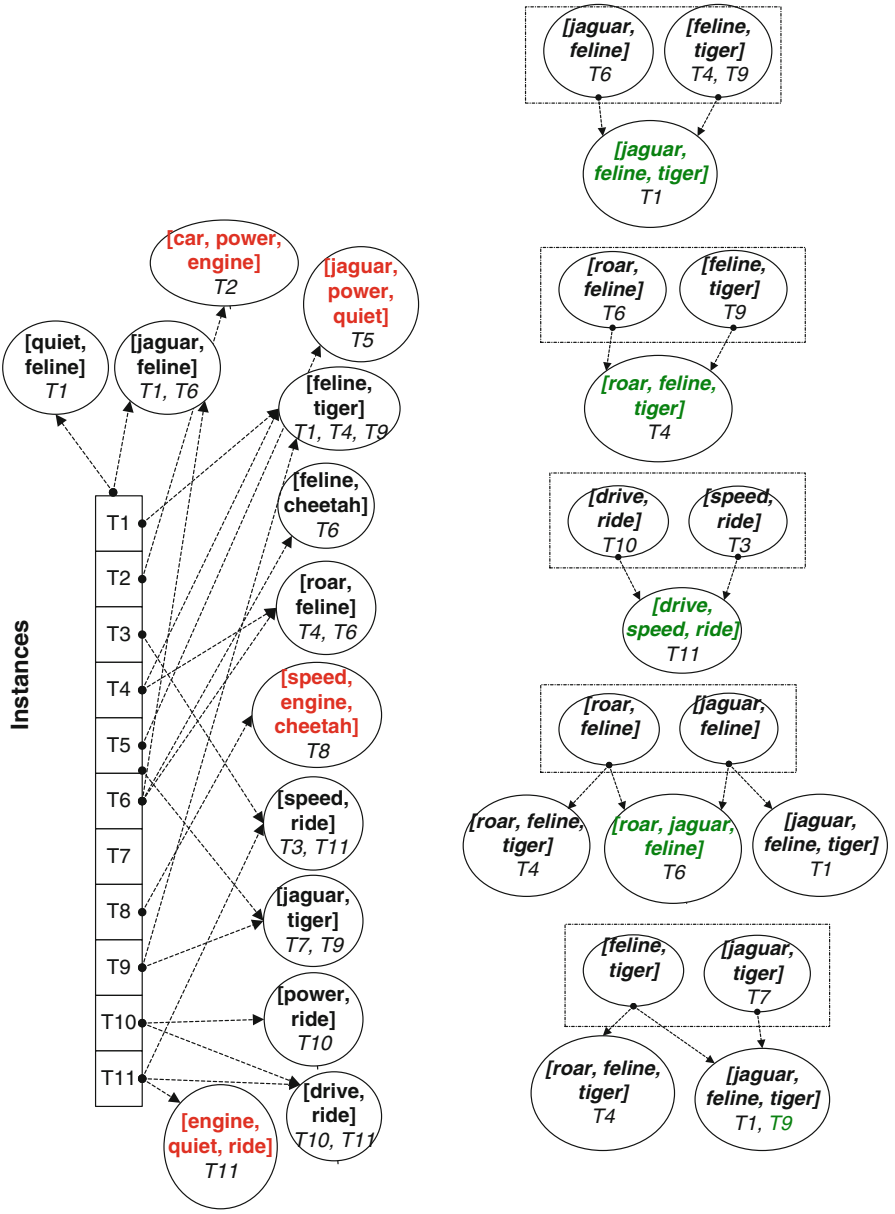

Fig. 2 Duplicate removal and cluster expansion. Newly added clusters and instances are lightly colored

Table 5 Datasets and their characteristics

Dataset	# Classes	# Instances	# Features
Reuters	90	10,787	19,127
Classic	4	7,094	41,681
Hitech	6	2,301	22,498
k1a	20	2340	21,839
k1b	6	2340	21,839
la12	6	6,279	30,125
mm	2	2521	126,373
Ohscal	10	11,162	11,465
re0	13	1,504	2,886
Reviews	5	4,069	36,746
Sports	7	8,580	27,673
tr11	9	414	6,429
tr12	8	313	5,804
tr23	6	204	5,832
tr31	7	927	10,128
Wap	20	1,560	8,460

pointers in the instance-to-cluster list for each instance is upper bounded by *maxK* or *topK* (or some proportion to it, depending on the level). Finally, the last step of merging top-level hierarchy nodes (Sect. 4.3) scales linearly because it uses bisecting *k*-means, which scales essentially linearly with the number of non-zero entries in the dataset (Malik and Kender 2006; Zhao and Karypis 2005).

7 Empirical evaluation

We conducted an extensive empirical evaluation of IDHC on 16 standard text datasets, 9 of which were also used in Malik and Kender (2006). Characteristics of these datasets are summarized in Table 5. With the exception of Reuters[3], we obtained all datasets from Cluto clustering toolkit (Karypis 2003). We compared IDHC with two state of the art algorithms, and evaluated effects of various parameters on IDHC performance, significantly expanding on our prior experimental work in Malik and Kender (2008).

We used the two standard hierarchical clustering evaluation metrics, FScore and entropy, as defined in Zhao and Karypis (2005), to compare the quality of cluster hierarchies produced by IDHC with two state-of-the-art hierarchical clustering algorithms. FScore combines the standard precision and recall functions commonly used in Information Retrieval, and evaluates the overall quality of hierarchical tree using a small number of its nodes. For each ground truth class (i.e., each unique class label assigned to any instance in the dataset), FScore identifies the node in the hierarchical

[3] http://kdd.ics.uci.edu/databases/reuters21578.

tree that best represents it and then measures the overall quality of the tree by evaluating this subset of clusters. Specifically, given a particular class L_r of size n_r and a particular cluster S_i of size n_i, suppose n_r^i documents in the cluster S_i belong to L_r, then the F value of this class and the cluster is defined as:

$$F(L_r, S_i) = \frac{2 * R(L_r, S_i) * P(L_r, S_i)}{R(L_r, S_i) + P(L_r, S_i)} \tag{2}$$

where $R(L_r, S_i) = n_r^i / n_r$ is the recall value and $P(L_r, S_i) = n_r^i / n_i$ is the precision value defined for the class L_r and the cluster S_i. It follows from these definitions that $F(L_r, S_i) \in [0, 1]$. The FScore of class L_r is the maximum F value attained at any node in the hierarchical tree T. That is,

$$\text{FScore}(L_r) = \max_{S_i \in T} F(L_r, S_i). \tag{3}$$

The FScore of the entire hierarchical tree is defined to be the sum of the individual class specific FScores weighted according to the class size. That is,

$$\text{FScore} = \sum_{r=1}^{c} \frac{n_r}{n} \text{FScore}(L_r). \tag{4}$$

where c is the total number of classes. FScore clearly lies in the interval [0, 1]. In general, higher FScore values indicate a better clustering solution. On the other hand, entropy takes into account the distribution of documents in all nodes of the tree. Given a particular node S_r of size n_r, the entropy of this node is defined to be:

$$E(S_i) = -\frac{1}{|\log(q)|} \sum_{i=1}^{q} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r} \tag{5}$$

where q is the total number of classes and is the number of documents of the ith class that were assigned to the rth node. Then, the entropy of the entire tree is:

$$E(T) = E(S_i) = \frac{1}{p} \sum_{i=1}^{p} E(S_i) \tag{6}$$

where p is the total number of non-leaf nodes of the hierarchical tree T. In general, lower entropy values indicate a better clustering solution. Like FScore, Entropy takes values in [0, 1].

As discussed in Sect. 4.1, we evaluated IDHC with two pattern selection schemas: a standard deviation based scheme and a top-k scheme. Unless otherwise specified, we used the fix parameter values of *minStDev*=1.5 and *maxK*=6 for the standard deviation based scheme and k=5 for the top-k scheme. This value for *minStdDev* was obtained by executing IDHC on one dataset (Reuters) and varying *minStdDev* between 1.0 and 4.0, in intervals of 0.1, selecting the value giving the best entropy score on the selected

dataset. Unless otherwise specified, all IDHC results presented here used Cluto post-processing (Sect. 4.3) and used a fixed value for the local coverage parameter, i.e., n=10.

7.1 Other algorithms

We selected bisecting k-means with I_2 criterion function (Zhao and Karypis 2005) as the first state-of-the-art algorithm for comparison. I_2 criterion function is defined as:

$$\text{maximize} \sum_{r=1}^{k} \sum_{d_i \in S_r} cos(d_i, C_r) \tag{7}$$

where C_r is a centroid of the r-th cluster S_r. Essentially, each cluster is represented by a centroid vector, and the idea is to find the solution maximizing the similarity between each document and the centroid of the cluster it is assigned to. As mentioned before (Sect. 4.3), bisecting k-means repeatedly partitions each cluster into two (trying to optimize the above criterion) until each cluster consists of only one document, producing a complete dendrogram. Therefore, no parameter for the number of clusters in needed. We also selected our previous global pattern-based hierarchical clustering algorithm, referred to as GPHC here as the second state-of-the-art algorithm for comparison with IDHC.

We limited our comparison to these two existing algorithms, since Malik and Kender (2006) showed that GPHC significantly outperformed FIHC (Fung et al. 2003) and TDC (Yu et al. 2004). Furthermore, Zhao and Karypis (2005) reported that bisecting k-means with I_2 criterion function outperforms UPGMA. Findings in Malik and Kender (2006) were also consistent with this observation. Consequently, we do not compare our algorithm against FIHC, TDC and UPGMA. In addition, we do not compare our algorithm against HICAP as it is reported Xiong et al. (2004) to have a performance that is comparable to UPGMA.

Like IDHC, GPHC requires an interestingness measure. We used MI (i.e., Mutual Information), which was found to be the top measure for it on text datasets in Malik and Kender (2006); see Sect. 7.8 for a note on the MI thresholds used for GPHC.

Bisecting k-means implementation from the Cluto clustering toolkit (Karypis 2003) was used. Since it relies on randomized heuristics, we attempted to ensure fairness by using the same dedicated machine (i.e., a 64-bit server with two Xeon processors and 8 GB of memory) to execute each clustering algorithm on each dataset 10 times and reported the averages. We note that Cluto produces both flat and hierarchical clustering solutions for bisecting k-means. The hierarchical solution is a dendrogram as discussed in Sect. 4.3. In contrast, the flat solution is obtained by applying secondary cluster analysis on this dendrogram (with number of clusters as a user defined parameter). We ensured an apples-to-apples comparison by always considering the full hierarchical solution (i.e., the dendrogram) and by using the same code to calculate FScore and entropy values for cluster hierarchies produced by all three algorithms.

Finally, since the iterative cluster refinement process in IDHC (Sect. 4.3) may produce many clusters containing only a single document leading to inflated IDHC entropy scores, we ensured fairness by excluding all IDHC clusters with a single document for computing FScore and entropy.

7.2 Evaluation tasks

In the following sections we will describe results of experiments designed to examine different aspects of IDHC performance relative to other methods and to choices of different parameters. Specifically, we perform experiments that:

- Compare IDHC with two state-of-the-art hierarchical classifiers, bisecting k-means and GPHC (Sect. 7.3), showing IDHC to be superior to both;
- Evaluate the choice of within-dataset interestingness measure for size-2 patterns (Sect. 7.4), with best overall results obtained using Added Value;
- Examine the effect of local significance measure and the cluster merging step on performance (Sect. 7.5). Use of local significance improves overall performance; while the merging step leads to better FScores but worse Entropy scores.
- Study the effect of varying local coverage, n, and minimum support $minS$ parameters (Sect. 7.6). These experiments show that for a range of reasonable parameter values (i.e. $n \in [5, 20]$ and $minS \in [2, 50]$) the performance of IDHC is rather stable, both with top-k and with standard deviation schemes.
- Study the effect of varying local coverage, n, and number of top patterns k, in top-k scheme (Sect. 7.7) showing that larger k leads to better entropies, but intermediate k leads to best FScores.
- We also show that IDHC uses significantly fewer patterns compared to methods that focus on global patterns, while having a better performance (Sect. 7.8).

7.3 Clustering performance

Tables 6 and 7 present the results of IDHC (both with the standard deviation and top-k schemes), bisecting k-means and GPHC. Figure 3 provides a visual summary of these tables. In order to demonstrate that IDHC is not very sensitive to parameter values, we used the same fixed parameter values on all 16 datasets. Tuning these parameters for each dataset separately may lead to slight further improvements.

From Table 6, we observe that GPHC achieved the highest FScores on 5 datasets and bisecting k-means achieved the highest FScores on 2 datasets, whereas the two IDHC pattern selection schemes achieved highest FScores on 9 datasets. Note that the two IDHC schemes generally performed similarly, and that their average performance was better than that of either GPHC or bisecting k-means. When IDHC performed worse than either of the other methods, the differences were relatively small (ex. classic, k1b, ohscal, mm). Frequently however, IDHC was significantly better than the other two methods (ex. tr* datasets). In terms of entropy (Table 7), GPHC and bisecting k-means were about 50% worse on average as compared to IDHC. These results convincingly demonstrate that IDHC is a better method overall.

Table 6 Clustering quality (FScores) on text datasets for 3 hierarchical clustering methods

Dataset	Bisecting *k*-means	GPHC	IDHC	
			minStdDev	top-*k*
Classic	0.782	**0.880**	0.749	0.814
Hitech	0.528	0.540	0.539	**0.542**
k1a	0.668	0.654	0.672	**0.677**
k1b	0.882	**0.903**	0.886	0.893
la12	0.741	0.661	0.746	**0.748**
mm	0.774	**0.943**	0.902	0.924
Ohscal	**0.601**	0.530	0.552	0.562
re0	0.610	**0.672**	0.623	0.623
Reuters	0.835	0.851	**0.855**	0.847
Reviews	0.801	0.818	0.837	**0.843**
Sports	0.882	**0.886**	0.874	0.883
tr11	0.795	0.519	0.791	**0.800**
tr12	0.689	0.604	**0.758**	0.739
tr23	0.667	0.487	0.718	**0.754**
tr31	0.837	0.584	**0.861**	0.823
Wap	**0.683**	0.663	0.662	0.674
Average	0.736	0.700	0.752	**0.759**

Bold numbers indicate the best result for each dataset

We also observe that the flexible overlapping clustering scheme adapted by IDHC is more beneficial to multi-label datasets as compared to single-label datasets. Among the datasets used in our experiments, Reuters was the only multi-label dataset and IDHC, when used with the standard deviation scheme, outperformed other algorithms with a substantially higher margin in terms of entropy (Table 7) on this dataset as compared to the average entropy improvement achieved on single-label datasets. This is because instances that dynamically select a higher number of size-2 patterns are more likely to belong to multiple topics and several instances that belong to each topic are likely to select common patterns, thus increasing their chances to share clusters with other instances that share at-least one class with them.

7.4 Interestingness measures

In this section, we compare the relative performance of top-6 interestingness measures (Geng and Hamilton 2006; Tan et al. 2002) to compute the within-dataset significance of size-2 patterns (as described in Sect. 4.1). Figure 4 presents the results for each of the two pattern selection schemes. We observe that on average Added Value (AV) is comparable to or slightly better than Yule's Q (YQ) and Certainty Factor (CF), with Mutual Information (MI) and χ^2 (CHI) performing somewhat worse than these measures and Conviction (V) performing much worse. Therefore, the rest of the results presented in this work used Added Value as the interestingness measure.

Table 7 Clustering quality (entropies) on text datasets for 3 hierarchical clustering methods

Dataset	Bisecting *k*-means	GPHC	IDHC	
			minStdDev	top-*k*
Classic	0.060	**0.025**	0.044	0.049
Hitech	0.224	0.172	**0.150**	0.182
k1a	0.106	**0.045**	0.068	0.083
k1b	0.042	0.042	**0.035**	0.041
la12	0.120	**0.062**	0.074	0.095
mm	0.073	0.053	**0.028**	0.041
Ohscal	**0.198**	0.237	0.204	0.222
re0	0.115	0.077	**0.031**	0.067
Reuters	0.075	0.156	**0.008**	0.026
Reviews	0.073	0.048	**0.029**	0.045
Sports	0.030	0.016	**0.012**	0.020
tr11	0.107	0.141	**0.076**	0.095
tr12	0.133	0.161	**0.074**	0.089
tr23	0.136	**0.042**	0.065	0.070
tr31	0.041	0.114	**0.025**	0.037
Wap	0.106	**0.047**	0.074	0.087
Average	0.102	0.090	**0.062**	0.078

Bold numbers indicate the best result for each dataset

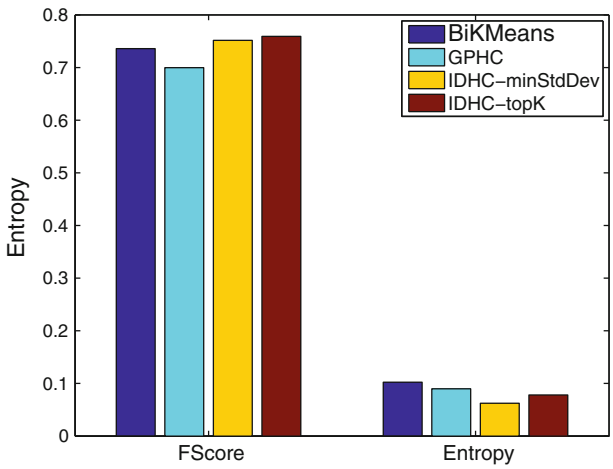

Fig. 3 FScores and entropies (averaged over all datasets) for bisecting *k*-means, GPHC and two IDHC schemes (*left* to *right*)

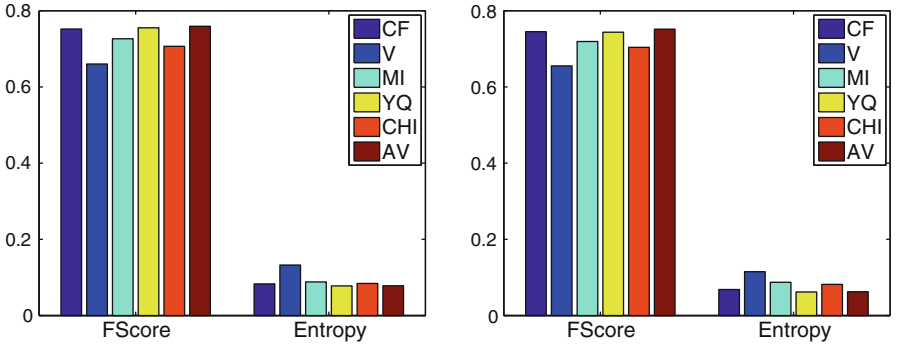

Fig. 4 Averages of FScores and Entropies across the datasets, obtained with different interestingness measures, for top-*k* (*left*) and standard deviation (*right*) schemes. Top-to-bottom on legend corresponds to left-to-right in the plot

7.5 Contribution of local significance and merging top-level clusters

In this section, we discuss two variants of IDHC to demonstrate the importance of local significance for pattern selection (Sects. 1.3 and 4.1) and of merging top-level clusters (Sect. 4.3). To demonstrate the importance of local pattern significance, we replaced line 7 of Algorithm 1 with $s_l = 1$ which results in ignoring local frequencies altogether. Similarly, we skipped top-level node merging by eliminating line 23 of Algorithm 1. For both variants, we executed IDHC on all datasets using default parameters. FScores and Entropies obtained with these variants, as well as the default IDHC results, are shown in Tables 8 and 9.

Without top-level node merging the entropies are much smaller as compared to all other variants (0.038 and 0.045), however, so are the FScores (0.185 and 0.178). This indicates that without this post-processing, IDHC creates very fine-grained clusters. This may be important for applications where cluster purity is very important, for example duplicate story detection.

Without local frequencies, IDHC still achieves decent FScores (averages of 0.734 and 0.728) - better than the GPHC result (average of 0.700). However, these results are noticeably worse than the default setting where local frequency information is used (averages of 0.752 and 0.759). The entropies without local frequencies are much worse than with the other approaches. This suggests that while within-dataset significance alone provides a lot of information, local pattern significance is important for achieving good clustering quality.

7.6 Effects of varying local coverage threshold and minimum support

Here we discuss the influence of the local coverage parameter n and minimum support *MinS*. Results with the standard deviation scheme are in Fig. 5 When *MinS* is low, all words that occur in more than 1 document are used, and so the local coverage parameter has no effect on the results. As *MinS* increases, fewer terms are included

Table 8 Clustering quality (FScores) on text datasets for different variants of IDHC, illustrating importance of top-level merging and of using local frequencies

	IDHC default		Without top-level node merging		Without local frequencies	
Dataset	*minStdDev*	top-*k*	*minStdDev*	top-*k*	*minStdDev*	top-*k*
Classic	0.749	0.814	0.056	0.065	0.849	0.767
Hitech	0.539	0.542	0.099	0.094	0.522	0.532
k1a	0.672	0.677	0.140	0.133	0.674	0.671
k1b	0.886	0.893	0.093	0.088	0.906	0.891
la12	0.746	0.748	0.136	0.129	0.732	0.732
mm	0.902	0.924	0.081	0.072	0.935	0.932
Ohscal	0.552	0.562	0.101	0.091	0.527	0.544
re0	0.623	0.623	0.311	0.315	0.585	0.584
Reuters	0.855	0.847	0.357	0.344	0.793	0.775
Reviews	0.837	0.843	0.116	0.106	0.817	0.824
Sports	0.874	0.883	0.166	0.153	0.860	0.855
tr11	0.791	0.800	0.235	0.222	0.803	0.770
tr12	0.758	0.739	0.272	0.268	0.660	0.669
tr23	0.718	0.754	0.331	0.319	0.641	0.623
tr31	0.861	0.823	0.308	0.298	0.820	0.832
Wap	0.662	0.674	0.157	0.149	0.628	0.646
Average	0.752	0.759	0.185	0.178	0.734	0.728

globally, so local coverage becomes more important. The worst results are observed with high *MinS* and $n = 0$, however, the results are rather stable elsewhere.

With the top-*k* scheme (Fig. 6), the observations are similar: high local coverage compensates for high minimum support. When *MinS* is low, local coverage has very little effect. Overall, the results tend to be slightly better and more sensitive to parameter values with the top-*k* scheme, however, these differences are not significant.

7.7 Effect of varying *k* in the top-*k* pattern selection scheme

In this section, we analyze the effect of varying *k* in the top-*k* pattern selection scheme. As shown in Fig. 7, for any value of the local coverage threshold (i.e., *n*), the entropies are lowest when *k* is high, while FScores are highest when *k* is not very high (3 or 5). Intuitively, when *k*=1 we don't get enough good features and the cluster overlap is minimal, but with much larger values of *k* we start getting features that aren't very useful and it results in too many instances to shares clusters with instances from other classes. It also appears that for a fixed value of *k* in the examined range, increasing *n* does not improve entropies and leads to some decline in the FScore (with an exception of *k*=5, where best FScore is achieved with *n*=20).

Table 9 Clustering quality (Entropies) on text datasets for different variants of IDHC, illustrating importance of top-level merging and of using local frequencies

	IDHC default		Without top-level node merging		Without local frequencies	
Dataset	*minStdDev*	top-*k*	*minStdDev*	top-*k*	*minStdDev*	top-*k*
Classic	0.044	0.049	0.011	0.027	0.076	0.055
Hitech	0.150	0.182	0.101	0.112	0.241	0.242
k1a	0.068	0.083	0.034	0.036	0.123	0.127
k1b	0.035	0.041	0.028	0.028	0.037	0.040
la12	0.074	0.095	0.049	0.058	0.120	0.133
mm	0.028	0.041	0.008	0.012	0.090	0.109
Ohscal	0.204	0.222	0.169	0.181	0.210	0.236
re0	0.031	0.067	0.033	0.049	0.122	0.125
Reuters	0.008	0.026	0.007	0.015	0.074	0.089
Reviews	0.029	0.045	0.012	0.016	0.074	0.090
Sports	0.012	0.020	0.009	0.013	0.029	0.038
tr11	0.076	0.095	0.043	0.048	0.130	0.131
tr12	0.074	0.089	0.034	0.039	0.177	0.180
tr23	0.065	0.070	0.023	0.015	0.151	0.152
tr31	0.025	0.037	0.019	0.029	0.070	0.074
Wap	0.074	0.087	0.035	0.041	0.132	0.138
Average	0.062	0.078	0.038	0.045	0.116	0.122

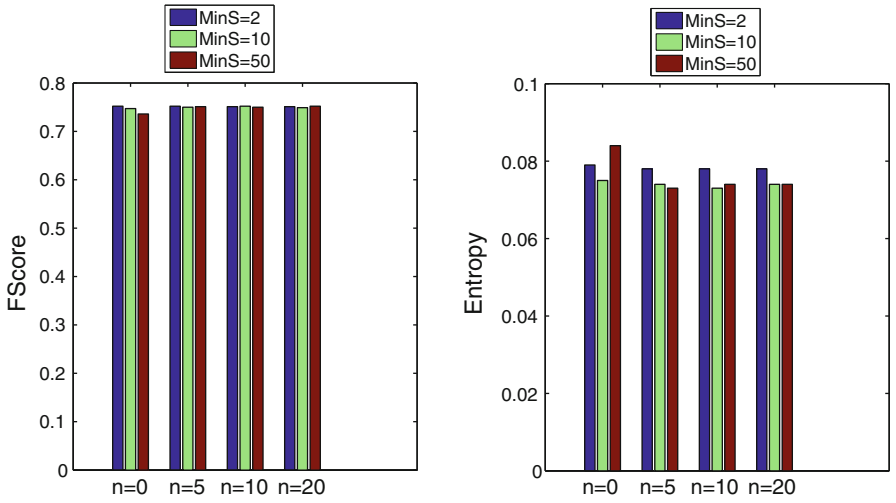

Fig. 5 FScores and entropies (averaged over all datasets) for different values of *n* and *MinS*, with standard deviation scheme. *Top*-to-*bottom* on legend corresponds to *left*-to-*right* in the plot

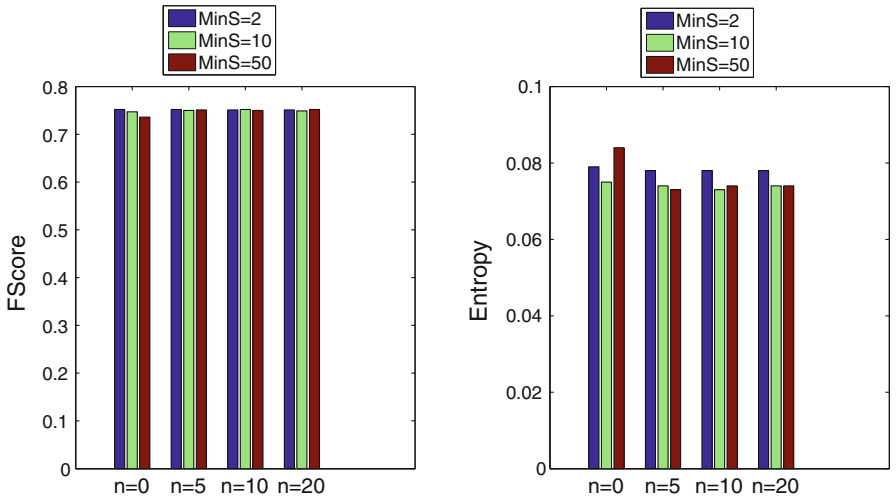

Fig. 6 FScores and entropies (averaged over all datasets) for different values of n and *MinS*, with top-k scheme. Top-to-bottom on legend corresponds to left-to-right in the plot

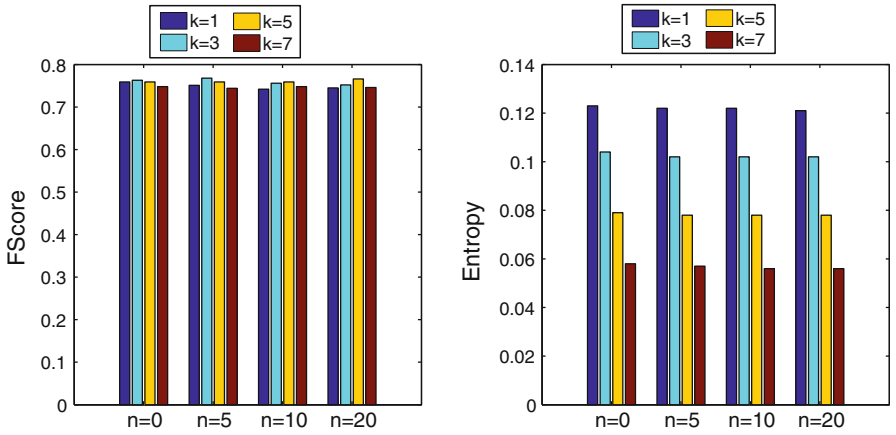

Fig. 7 FScores and entropies (averaged over all datasets) for different values of n and k using the top-k scheme. Top-to-bottom on legend corresponds to left-to-right in the plot

7.8 Robustness in the number of patterns

In Sect. 1.2, we noted that the threshold-based global pattern mining step in existing algorithms may result in an unpredictable number of patterns. During our experiments, we found many examples that clearly demonstrate this problem. We provide a few here and also show that IDHC does not suffer from this problem.

In Malik and Kender (2006), we showed that the GPHC algorithm worked well on all 9 datasets used there with interestingness threshold values of 0.1 and 0.85 for MI and YulesQ, respectively. However, using the same threshold values on some of the highly correlated datasets leads to an extremely large number of size-2 patterns

Table 10 Number of size-2 patterns for GPHC (Malik and Kender 2008) and IDHC (using the standard deviation scheme and default parameters)

Dataset	Approximate # of size-2 patterns		
	GPHC, MI (million)	GPHC, YulesQ	IDHC
mm	2.4	Fails	3, 651
Reviews	2.6	Fails	5, 952
Sports	1.4	Fails	12, 607
tr11	4.3	11.4 million	604
tr12	3.6	8.8 million	464
tr23	7.6	12.2 million	282
tr31	7.0	Fails	1, 360

IDHC uses significantly fewer patterns while producing better results

(Table 10), so that in a number of cases the mining process could not continue because it has exhausted the available system resources. In some cases, GPHC could not even finish mining size-2 patterns. Thus, we had to increase threshold values to obtain the results reported in Sect. 7.3.

The problem of generating too many patterns is not unique to GPHC. All global pattern mining based clustering algorithms suffer from a similar problem. In fact, a pure frequent or closed frequent itemset based algorithm would be expected to find an even higher number of itemsets (Malik and Kender 2006). In contrast, our local standard deviation based pattern selection scheme selected up to three orders of magnitude fewer size-2 patterns, without loss of performance in general (Sect. 7.3). If the top-k scheme is used, the maximum number of size-2 patterns selected for each instance can be directly controlled using the k parameter.

8 Conclusions and future work

IDHC is an instance of the LeGo framework that uses local patterns as features for constructing a cluster hierarchy. These patterns are mined using an instance-driven approach that selects representative size-2 patterns for each instance in the dataset. This eliminates the need to use highly unstable global thresholds and also guarantees that the final set of selected patterns covers all instances. The selected patterns form initial clusters, described by sets of local patterns, and a cluster hierarchy is obtained by following an iterative cluster refinement process that avoids global processing by utilizing instance-to-cluster relationships, produces more meaningful cluster labels, and allows for generating more flexible overlapping clusters.

Our extensive experimental evaluation on 16 datasets shows that the proposed approach outperforms existing hierarchical clustering algorithms both in terms of FScore and entropy. Most importantly, this approach was equally effective on highly correlated datasets, while using a lot fewer patterns than other methods. These results clearly demonstrate the effectiveness of LeGo approach and show that global models constructed using local patterns may substantially outperform global models constructed

using globally significant patterns, even if local patterns were minded using a rather simple method. We also demonstrate the importance of individual steps in the IDHC algorithm, such as use of local significance and the merging of clusters in the final stages. While the algorithm relies on several parameters, such as minimum support $minS$ and local coverage n, for a large range of values of these parameters performance is very similar, indicating that the proposed algorithm is rather stable. The use of Added Value as a pattern significance measure was experimentally supported in comparison with 5 other measures.

In the future, we plan to investigate more sophisticated ways of calculating local pattern significance and incorporate methods from natural language processing that may help restrict the local pattern search space even further. In addition, we plan to apply IDHC on non-textual dense datasets such as the UCI dataset collection.

Acknowledgements We would like to thank the editors of this special issue and the anonymous reviewers for their constructive and detailed comments. We would also like to thank Professor Howard Hamilton who reviewed an earlier version of this paper and provided valuable feedback.

References

Angiulli F, Ianni G, Palopoli L (2001) On the complexity of mining association rules. In: Proceedings of Nono Convegno Nazionale su Sistemi Evoluti di Basi di Dati (SEBD), pp 177–184

Beil F, Ester M, Xu X (2002) Frequent term-based text clustering. In: Proceedings of international conference on knowledge discovery and data mining, pp 436–442

Brijs T, Vanhoof K, Wets G (2003) Defining interestingness for association rules. Int J Inf Theor Appl 10(4):370–376

Carter C, Hamilton H, Cercone N (1997) Share based measures for itemsets. In: Proceedings of the first European symposium on principles of data mining and knowledge discovery, pp 14–24

Clifton C, Coolie R, Rennie J (2004) TopCat: Data mining for topic identification in a text corpus. IEEE Trans Knowl Data Eng 16(8):949–964

Fung B, Wang K, Ester M (2003) Hierarchical document clustering using frequent itemsets. In: Proceedings of SIAM international conference on data mining, pp 59–70

Geng L, Hamilton HJ (2006) Interestingness measures for data mining: a survey. ACM Comput Surv 38(3). http://portal.acm.org/citation.cfm?id=1132960.1132963

Gunopulos D, Khardon R, Mannila H, Saluja S, Toivonen H, Sharma RS (2003) Discovering all most specific sentences. ACM Trans Database Syst 28(2):140–174

Han E.H., Karypis G, Kumar V, Mobasher B (1997) Clustering based on association rule hypergraphs. In: Proceedings of research issues on data mining and knowledge discovery, pp 59–70

Karypis G (2003) CLUTO: A software package for clustering high dimensional datasets. http://www-users.cs.umn.edu/~karypis/cluto/

Knobbe A, Crémilleux B, Fürnkranz J, Scholz M (2008) From local patterns to global models: the LeGo approach to data mining. In: Proceedings of local patterns to global models workshop (ECML/PKDD), pp 1–16

Li Y, Chung S. M, (2005) Text document clustering based on frequent word sequences. In: Proceedings of the 14th ACM international conference on information and knowledge management (CIKM), pp 293–294

Malik H, Kender JR (2006) High quality, efficient hierarchical document clustering using closed interesting itemsets. In: Proceedings of sixth IEEE international conference on data mining, pp 991–996

Malik H, Kender JR (2007) Optimizing frequency queries for data mining applications. In: Proceedings of Seventh IEEE International Conference on Data Mining, pp 595–600

Malik H, Kender JR (2008) Instance driven hierarchical clustering of document collections. In: Proceedings of local patterns to global models workshop (ECML/PKDD)

Moerchen F, Brinker K, Neubauer C (2007) Any-time clustering of high frequency news streams. In: Proceedings of data mining case studies workshop, the thirteenth ACM SIGKDD international conference on knowledge discovery and data mining

Parsons L, Haque E, Liu H (2004) Subspace clustering for high dimensional data: a review. ACM SIGKDD Explor Newslett 6(1):90–105

Salton G, Buckley C (1988) Term-weighting approaches in automatic text retrieval. Inf Process Manag 24((5):513–523

Tan P, Kumar V, Sristava J (2002) Selecting the right interestingness measure for association patterns. In: Proceedings of 8th international conference on knowledge discovery and data mining, pp 32–41

Wang J, Karypis G (2004) SUMMARY: efficient summarizing transactions for clustering. In: Proceedings of fourth IEEE international conference on data mining, pp 241–248

Wu C (2006) Mining top-K frequent closed itemsets is not in APX. In: Proceedings of PAKDD, pp 435–439

Xiong H, Steinbach M, Tan PN, Kumar V (2004) HICAP: hierarchical clustering with pattern preservation. In: Proceedings of SIAM international conference on data mining, pp 279–290

Yang Y, Pedersen JO (1997) A comparative study on feature selection in text categorization. In: Proceedings of the 14th international conference on machine learning, pp 412–420

Yu H, Searsmith D, Li X, Han J (2004) Scalable construction of topic directory with nonparametric closed termset mining. In: Proceedings of fourth IEEE international conference on data mining, pp 563–566

Zhao Y, Karypis G (2005) Hierarchical clustering algorithms for document datasets. Data Min Knowl Discov 10(2):141–168