Classification by Pattern-Based Hierarchical Clustering

Hassan H. Malik, and John R. Kender

Department of Computer Science, Columbia University,
New York, NY 10027, USA
{hhm2104, jrk}@cs.columbia.edu

Abstract. In this paper, we propose CPHC, a semi-supervised classification algorithm that uses a pattern-based cluster hierarchy as a direct means for classification. All training and test instances are first clustered together using an instance-driven pattern-based hierarchical clustering algorithm that allows each instance to "vote" for its representative size-2 patterns in a way that balances local pattern significance and global pattern interestingness. These patterns form initial clusters and the rest of the cluster hierarchy is obtained by following a unique iterative cluster refinement process that exploits local information. The resulting cluster hierarchy is then used directly to classify test instances, eliminating the need to train a classifier on an enhanced training set. For each test instance, we first use the hierarchical structure to identify nodes that contain the test instance, and then use the labels of co-existing training instances, weighing them proportionately to their pattern lengths, to obtain the most likely class label(s) for the test instance. In addition, CPHC increases the chances of classifying isolated test instances by inducing a type of feature transitivity. Results of experiments performed on 19 standard text and machine learning datasets show that CPHC outperforms a number of existing classification algorithms even with sparse (as low as 1%) training data.

Keywords: Semi-supervised classification, pattern-based hierarchical clustering, transductive learning, interestingness measures.

1 Introduction

Traditional inductive classifiers are trained on instances (Section 1.1) in the training set to produce a classification model (or knowledge base). This model is later used to classify previously unseen test instances. Considering that these classifiers may not fully exploit the distribution of test instances in the context of the whole dataset (i.e., by building the classification model only from the training instances, while ignoring test instances altogether), a number of recent approaches [14, 23, 34] adopted a semi-supervised model for classification. These approaches first apply an unsupervised, flat clustering algorithm (i.e., *k*-means clustering) to cluster all (i.e., training and test) instances in the dataset, and then use the resulting clustering solution to add additional instances to the training set. A classifier is then trained on the enhanced training set.

However, the quality of clustering achieved by traditional flat clustering algorithms (i.e., *k*-means clustering) relies heavily on the desired number of clusters (i.e., the value of *k*), which must be known in advance. Unfortunately, setting a good value for *k* can be non-trivial and no successful methods exist to automatically determine this value for a new, previously unseen dataset. Therefore, flat clustering 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. This also means that an inappropriate value for *k* may adversely impact the quality of classification achieved by existing semi-supervised classification algorithms [14, 23, 34].

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 include agglomerative algorithms such as UPGMA [35] and partitioning based algorithms such as bisecting *k*-means [35]. Additionally, a number of pattern-based hierarchical clustering algorithms have achieved success on a variety of datasets [3, 10, 18, 31, 33].

Traditional agglomerative and partitioning-based hierarchical clustering algorithms merge exactly two nodes at each step, which may result in a "mechanical looking" hierarchy that may not resemble hierarchies produced by human experts. In addition, these algorithms do not automatically generate cluster labels, and do not support soft clustering. In contrast, pattern-based hierarchical clustering algorithms allow each node in the cluster hierarchy to have a variable number of child nodes, which may in general be closer to a real-life setting. Pattern-based hierarchical clustering algorithms also automatically generate cluster labels (i.e., the set of binary attributes defining each cluster), and more easily support soft clustering (i.e., a technique that assigns instances to one or more clusters).

Considering these advantages, we propose CPHC (i.e., **C**lassification by **P**attern-based **H**ierarchical **C**lustering), a novel semi-supervised classification algorithm that uses a pattern-based cluster hierarchy as a direct means for classification. Unlike existing semi-supervised classification algorithms, CPHC directly uses the resulting cluster hierarchy to classify test instances and hence eliminates the extra training step.

The remainder of this section briefly introduces the notations used in this paper, discusses the motivation for instance-driven pattern-based hierarchical clustering, discusses the significance of pattern lengths in these hierarchies and also provides a brief overview of the CPHC algorithm. Section 2 summarizes existing work that is related to this research. Section 3 provides details on various steps in our classification process. Section 4 compares the performance of CPHC against state-of-the-art machine-learning and data-mining-based classification algorithms. Finally, we conclude and discuss ideas for future work in Section 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, and $C = \{c_1, c_2, c_3, \ldots, c_m\}$ be the complete set of distinct class labels. An instance X is denoted as a triple $<id, L, Y>$ such that *id* is an identifier that uniquely identifies X, $L \subseteq C$ represents the set of class labels associated with X ($L=\Phi$ if X represents a test instance), and $Y \subseteq I$ 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 $D_P = \{(id, L, Y) \in D \,|\, P \subseteq Y\}$. The *support* of a pattern P is defined as:

$$Support(P) = \frac{|D_P|}{|D|}$$

1.2 An Alternative to Global Pattern-based Hierarchical Clustering

Most of the existing pattern-based hierarchical clustering algorithms [3, 10, 18, 31, 33] follow a similar framework. These algorithms first mine a set of globally significant patterns (e.g., frequent itemsets [3, 10], closed frequent itemsets [33], high h-confidence itemsets [31], or closed interesting itemsets [18]), and then use these patterns to build a cluster hierarchy. Each 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. We identified four major problems with existing pattern-based hierarchical clustering algorithms.

First, these algorithms use a global user-defined threshold (e.g., minimum support or minimum h-confidence) to prune an exponentially large search space, and to obtain the final set of globally significant patterns used for clustering. Similar to the problem of setting the value of *k* in flat clustering, setting a suitable value for this threshold can be non-trivial. An inappropriate threshold value may result in too many or too few patterns, with no coverage guarantees (i.e., some instances might not contain any globally significant pattern).

Second, global pattern-mining algorithms (i.e., APRIORI [2], CLOSET+ [29], Closed Interesting Itemset mining [18]) used by the existing pattern-based clustering algorithms [3, 10, 18, 31, 33] only consider the presence or absence of patterns in instances, and ignore within-instance pattern significance. This may be inappropriate for real-life text and web datasets, where instances 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.

Third, existing pattern-based clustering algorithms [14, 23, 34] tightly couple the sizes of cluster labels with the node heights in the initial cluster hierarchy. In these approaches, 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).

Finally, many of the existing pattern-based hierarchical clustering algorithms apply artificial constraints on soft clustering. Some of these algorithms [18, 33] require the user to provide the number of clusters for each instance, and always select the maximum number of clusters whenever possible for each instance. Similarly, some of these algorithms [18, 33] only assign instances to their most specific pattern clusters.

In an attempt to address these issues, the authors have recently proposed IDHC [19], an instance-driven approach to pattern-based hierarchical clustering, which we review here. Instead of following the usual framework (i.e., first mining globally significant patterns and then using these patterns to build the cluster hierarchy), IDHC allows each instance to select a variable number of representative size-2 patterns in a way that ensures an effective balance between local and global pattern significance. The local (i.e., within instance) frequencies of the two items in a size-2 pattern are averaged together to obtain the local pattern significance, and a contingency table-based interestingness measure [12, 27] is used to obtain the global pattern significance. These local and global significance scores are then multiplied to obtain the overall pattern significance score with respect to the current instance. The patterns are sorted with respect to their significance scores, and the instance selects a subset of these patterns, the number of which is dynamically determined using a standard deviation based scheme. This scheme selects up to *maxK* patterns with significance scores that are greater than or equal to "*min_std_dev*" standard deviations from the mean, where *maxK* and *min_std_dev* are user definable parameters. Since there is no global pattern mining step, a global threshold is not needed. Furthermore, the total number of size-2 patterns is guaranteed to be linear in the total number of instances in the dataset, and all instances are guaranteed to be covered.

Once size-2 patterns are selected for all instances, each unique size-2 pattern forms a top level cluster in the hierarchy, and instances are associated with all the pattern clusters that they had selected, while maintaining instance-to-cluster pointers. These initial clusters are then refined to obtain the rest of the cluster hierarchy by following a novel iterative instance-driven process that simultaneously grows patterns and clusters, and also inherently avoids combinatorial explosion.

In each iteration, this process utilizes instance-to-cluster pointers to prune duplicate clusters in a purely local way (i.e., by only comparing cluster pairs that co-exist in an instance's list of cluster pointers). The labels of identified duplicates are merged and assigned to the retained cluster. Clusters are refined in a similar fashion by first identifying non-atomic cluster pairs (i.e., clusters that contain at least two instances) from each instance's list of cluster pointers, and then using these pairs to generate clusters for the next level. The newly generated clusters contain instances that are common to the originating cluster pair, and are assigned labels that represent the union of labels of both clusters in the originating cluster pair. The instance-to-cluster relationships are reestablished, and the refinement iteratively continues until all clusters are refined. Finally, since pattern-based cluster hierarchies may contain a large number of top level nodes, bisecting *k*-means (with I_2 criterion function [35]) is applied to merge these nodes.

This process produces more descriptive cluster labels than previous approaches, without tightly coupling node label sizes with node heights in the initial cluster hierarchy. In addition, this process does not force instances to their longest pattern clusters and allows each instance to exist at multiple levels in cluster the hierarchy.

See Figure 3 for an example. Results in [19] show that IDHC outperforms existing agglomerative, partitioning-based, and pattern-based hierarchical clustering algorithms both in terms of FScore and entropy [35].

1.3 The Significance of Pattern Lengths in Pattern-based Cluster Hierarchies

The two most common metrics used to evaluate the quality of cluster hierarchies are entropy and FScore. As noted in [35], entropy considers the distribution of instances in all nodes of the tree whereas FScore only considers one (best) node for each ground truth class, and ignores the quality of all other nodes. This means that a cluster hierarchy with better (i.e., lower) entropy is expected to have a higher percentage of nodes that contain most instances that belong to the same ground truth class. We performed further experiments to analyze the class-label distributions over nodes with varying pattern-lengths.

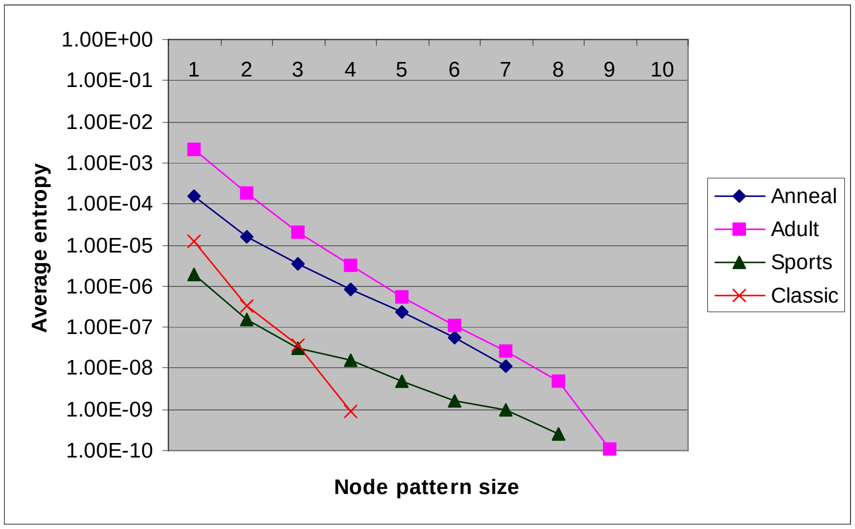

Fig. 1. Average entropies of nodes with respect to their pattern sizes on anneal, adult, sports and classic datasets. Note that pattern size = 1 represent "logical" nodes obtained by applying bisecting *k*-means to merge top-level nodes in the initial cluster hierarchy, as discussed in Section 1.2. Least-squares regression confirmed that the relationship is essentially linear.

Intuitively, since the IDHC algorithm (Section 1.2) only assigns instances to nodes (i.e., clusters) that represent their statistically selected patterns, we expected nodes with longer patterns to have lower entropies. To validate our intuition, we applied the IDHC algorithm to cluster two common machine learning datasets and two common text datasets. We calculated the resulting individual node entropies, and grouped together nodes that represented the same pattern sizes. We report average entropies of each group in semi-log format in Figure 1. We observe that average node entropies decreased (i.e., improved) monotonically and nearly linearly with increasing pattern-sizes on all four datasets, confirming our intuition.

1.4 CPHC: A Novel Classification Algorithm

Motivated by this observation, we now propose in this paper CPHC (i.e., **C**lassification by **P**attern-based **H**ierarchical **C**lustering), a novel semi-supervised classification algorithm that uses pattern-lengths as a way of establishing cluster (i.e., node) weights. CPHC first applies an unsupervised instance-driven pattern-based hierarchical clustering algorithm (i.e., IDHC, Section 1.2) to the whole dataset to produce a cluster hierarchy. Unlike existing semi-supervised classification algorithms [14, 23, 34], CPHC directly uses the resulting cluster hierarchy to classify test instances and hence eliminates the extra training step. To classify a test instance, CPHC first uses the hierarchical structure to identify nodes that contain the test instance, and then uses the labels of co-existing training instances, weighing them by node pattern-lengths (i.e., by multiplying the node pattern-interestingness value with the pattern-length) to obtain class label(s) for the test instance. This allows CPHC to classify unlabeled test instances without making any assumptions about their distribution in the dataset.

With results of experiments performed on 19 standard datasets, we show in Section 4 that CPHC outperforms a number of existing classification algorithms such as FindSim, Naïve Bayes, BayesNets, Trees, ARC-BC, FOIL and CPAR, and achieves classification accuracies that are comparable to (or better than) SVM and Harmony. Most importantly, CPHC was effective even with sparse (as low as 1%) training data.

The main contributions include: 1) A novel semi-supervised classification algorithm that uses a unified pattern-based cluster hierarchy as a direct means for classification, 2) A novel feature selection method that ensures that all training and test instances are covered by the selected features, 3) Elimination of the need to train any classifier on the enhanced training set and 4) Utilization of pattern-lengths to determine cluster (i.e., node) weights.

2 Related Work

Our work relates to existing rule and pattern-based classification algorithms, with several important differences. Rule-induction-based classifiers like FOIL [22], RIPPER [7], CPAR [32] and C4.5 [21] use heuristics such as *Gini Index* and *Information Gain* (or *Information Gain* variants), to identify the best literal by which to grow the current rule [30]. Many of them follow the sequential covering paradigm. In contrast, association rule-based classifiers such as CBA [17], CAEP [8], CMAR [16], ARC-BC [1], and DeEPs [15] first mine a large set of association rules that satisfy user-defined support and confidence thresholds, and then extract the final set of classification rules by following a database covering technique. With Harmony [30], Wang and Karypis proposed an instance-centric approach to mine classification rules. Harmony builds the classification model by directly mining some user-defined number of highest-confidence rules for each training instance that satisfy minimum support. Furthermore, Harmony simultaneously mines rules for all classes.

Our work also relates to a number of recently proposed approaches that use clustering as a way of enhancing the training set. We mention only a few of those approaches here. Raskutti et al. [23] used unlabeled data that is not part of the test set to improve the performance of text classification. This is achieved by clustering labeled and unlabeled instances together, and extracting new features from these clusters to enhance the classification model. In another approach, Zeng et al. [34] first clustered training and test sets together. The resulting clustering solution is then used to obtain labels for some of the unlabeled test instances, and the newly labeled instances are added to the training set. The extended training set is finally used to train a classifier. In a similar approach [14], Kyriakopoulou and Kalamboukis first clustered training and test sets together. The dataset is then augmented with meta features extracted from the resulting clusters, and a classifier is trained on the expanded dataset. In addition, a number of approaches like [20, 25] used clustering as a way of improving the feature selection for classification. These semi-supervised classification algorithms are similar to transductive learning [28] in that transductive learning also allows the structure of the test set to play a role in classification.

Our CPHC algorithm is similar to existing pattern-based classification algorithms in that we also use patterns. But unlike these algorithms, we do not attempt to construct a classification model from the training set. Our approach also differs from existing semi-supervised classification algorithms in that we do not use clustering as a way of enhancing the training set. Instead, we directly utilize a cluster hierarchy to classify test instances and therefore, avoid the extra step of training a classifier after clustering. In addition, existing approaches do not use pattern lengths as a way of establishing cluster weights.

```
Step 1: Select features (Section 3.1)
Input:  training instances $trn_1..trn_n$
        test instances $tst_1..tst_m$
            Select features as explained in Section 3.1
Output: $trn'_1..trn'_n$, and $tst'_1..tst'_m$ with reduced features
Step 2: Obtain a cluster hierarchy of training and test instances (Section 3.2)
Input:  training instances $trn'_1..trn'_n$
        test instances $tst'_1..tst'_m$
            Apply the IDHC algorithm (Section 1.2) on ($trn'_1..trn'_n$ U $tst'_1..tst'_m$)
Output: cluster hierarchy $h$
Step 3: Classify test instances (Section 3.3)
Input:  cluster hierarchy $h$
        test instances $tst_1..tst_m$
            For each test instance $tst_i$,
                Traverse $h$ from root to leaves, identify set $S$ of clusters that contain $tst_i$
                Use clusters in $S$, and lengths of associated patterns as their weights to compute
                class scores (i.e., by multiplying the node pattern-interestingness value with the
                pattern-length)
                Assign the label of top-scoring class (or classes for multi-label problems) to $tst_i$
Output: predicted labels of test instances $tst_1..tst_m$
```

Fig. 2. The CPHC Algorithm.

3 The CPHC Algorithm

In this Section, we explain various steps involved in the CPHC algorithm. Figure 2 summarizes these steps, and subsections 3.1-3.3 provide details on each step.

3.1 Step 1: Noise Elimination and Feature Selection

Studies [11, 24] show that reducing the dimensionality of the feature space may significantly improve the effectiveness and scalability of traditional classification algorithms, especially on high-dimensional datasets. Furthermore, dimensionality reduction tends to reduce overfitting [24]. Pattern-based classification algorithms equally benefit from dimensionality reduction, as both the quality and the number of discovered patterns directly depends on the number of initial items.

Typically, features are selected by first sorting all available features in terms of their significance, and then selecting top-*j*, or top-*j*-percent features (with a caveat that selecting a suitable value for *j* is not straightforward). A recent study [11] evaluated various measures to calculate feature significance and concluded that *Information Gain*, *Chi-Square* and *Bi-normal Separation* worked equally well on a number of datasets, with no statistically significant difference. Considering the comparatively high computational cost of common feature selection methods, a recent hidden-web classification algorithm [13] adopted an efficient, two-phase approach. In its first phase, Zipf's law was applied as an inexpensive heuristic dimensionality reduction technique to eliminate too frequent and too rare features. In its second phase, a more expensive method was applied to select the final set of features.

Unfortunately, none of these approaches guarantee coverage (i.e., that each instance in the corpus is represented by the selected features). Furthermore, the optimal number (or percentage) of features (i.e., the value of *j*) needed to achieve good classification results remains unclear. The literature [24] is inconclusive on *n*: some studies suggest that the number of selected features should be same as the number of training examples, and others suggest that feature selection may make matters worse, especially when the number of available features is small.

Since CPHC first produces a cluster hierarchy of the whole dataset, using a supervised feature selection method (i.e., *Information Gain*) alone may leave some test instances unrepresented in the cluster hierarchy. That is some test instances entirely consist of features that do not exist in any training instance. Traditional classification algorithms may not be able to classify such test instances at all. CPHC however, improves the chances of classifying such test instances by inducing a type of feature transitivity: as long as these isolated test instances share some features with more common test instances that overlap the training set, they have a chance of being clustered together in a "logical" node (see Section 3.3 for details).

Considering these issues, we adopt a heuristic feature selection method that is efficient, and ensures that the final set of selected features covers all training and test instances. Furthermore, using the number of training instances, and the number of available features, our method automatically estimates the number of features used for classification (i.e., the value of *j*). Our method consists of the following four steps:

***Step 1.1 (calculate j)*:** $j = \begin{cases} f & f < n \\ n + \left(n \times \log \dfrac{f}{n} \right) & otherwise \end{cases}$

where n = number of training instances, and f = total number of available features. This empirically derived formula ensures a reasonable base amount for low dimensional datasets, while moderately growing this number for high dimensional datasets.

***Step 1.2 (select globally significant features)*:** Heuristically select globally most useful features by first applying Zipf's law to select features that are neither too frequent, nor too infrequent (as these features are considered to be less significant). In other words, select features that exist in less than *max_supp*, and more than *min_supp* percent instances (where *min_supp* and are *max_supp* are user defined parameters). Further refine these features by first sorting them in decreasing order of their *Information Gain* values (computed using labeled training instances only), and then adding the resulting top-*j* features to set *S* (i.e., the set of "selected" features).

***Step 1.3 (ensure local coverage of training instances)*:** For each instance *X* in the training set represented as triple <*id*, *L*, *Y*> (Section 1.1), check if $|Y \cap S| \geq t$ (where *t* is user defined). If the condition is not met, sort all features in the current instance in the decreasing order of their (TF * *Information Gain*), where TF = local term frequency count in the instance. This "balances" the local significance (i.e., TF) and the global significance (i.e., *Information Gain*). Finally, add the resulting top ($t - |Y \cap S|$) features to set *S*.

***Step 1.4 (ensure local coverage of test instances)*:** For each instance *X* in the test set represented as triple <*id*, *L*, *Y*>, check if $|Y \cap S| \geq t$. If the condition is not met, sort all features in the current instance in the decreasing order of their Term Frequency values. Finally, add the resulting top ($t - |Y \cap S|$) features to set *S*.

3.2 Step 2: Hierarchical Clustering of Training and Test Instances

Once we have the features selected, we apply the IDHC algorithm (Section 1.2) on the whole dataset to obtain a cluster hierarchy. The IDHC algorithm computes interestingness values for selecting size-2 patterns for instances. However, in the original algorithm these values are not stored, since cluster refinement was done solely using instance-to-cluster pointers. But here, we need to use these values to calculate class scores for test instances (Section 3.3), so we modified the IDHC algorithm to track these values. In addition, we obtain interestingness values for patterns longer than size-2 by averaging the interestingness values of patterns merged during cluster refinement (Section 1.2). We also use the same process in a bottom-up fashion to obtain interestingness values for "logical" nodes (i.e., clusters) generated by merging the top-level nodes.

3.3 Step 3: Classifying Test Instances

We use the following four-step process to classify test instances:

***Step 3.1*:** Given a test instance *t*, and hierarchy *h*, first initialize scores for all classes. Next, traverse *h* from root to leaves, identifying the set *S* of nodes that contain *t*.

***Step 3.2*:** For each node *e* in *S*, compute *w* such that:

$$w = \text{node-pattern-length} * \text{node-interestingness}$$

This weight is based on the relationship presented in Figure 1.

***Step 3.3*:** For each class *c* represented by at least one training instance in *e* (considering all instances in the node as well as instances in all child nodes, as usual), add *x* to the score of *c* such that:

$$x = w \times \frac{|\text{training instances with label} = c \text{ in } e|}{|\text{training instances in } e|}$$

***Step 3.4*:** For single-label classification problems, select the label of the class with the highest score. For multi-label problems, select multiple classes using the "weighted dominant factor-based" scheme in Section V(C-3) of [30], except replacing all uses of confidence with the selected interestingness measure.

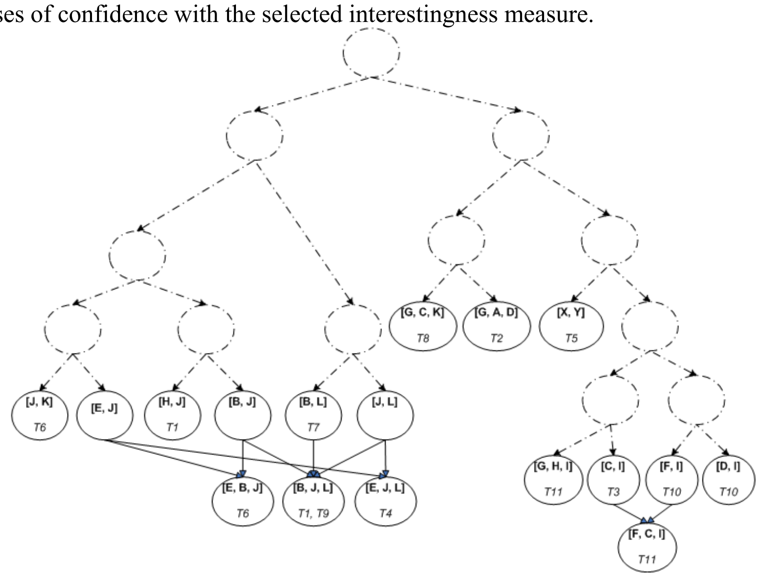

Fig. 3. A pattern-based cluster hierarchy obtained by applying the IDHC algorithm. The dotted nodes are "logical" nodes obtained by applying bisecting *k*-means to merge the top-level nodes in the initial cluster hierarchy.

Since traditional inductive classifiers only use features in training instances to obtain the classification model, these algorithms may not be able to classify test instances that entirely consist of features that do not exist in any training instance, even if these isolated test instances share some features with more common test instances that overlap the training set. CPHC improves the chances of classifying such test instances by inducing a type of feature transitivity: as long as these isolated test instances share some features with more common test instances that overlap the training set, they have a chance of being clustered together in a "logical" node (i.e., node obtained by merging top-level nodes in the initial cluster hierarchy; Section 1.2). As a result, the "logical" node may contribute towards score calculation.

Example: Figure 3 presents a pattern-based cluster hierarchy obtained by applying the IDHC algorithm (Section 1.2). Let us assume that T3 and T5 are test instances that share some features (i.e. feature "X"), and the remaining instances are training instances. Let us also assume that T5 is an "isolated" test instance, i.e., T5 does not share any features with the training set. Since T5 shares some features with T3 (i.e., a test instance that overlaps with the training set), T3 and T5 are clustered together in the logical node formed by merging the node with pattern "X, Y", and a logical node that contains T3. This structure allows the parent of node with pattern "X, Y" to predict class labels for T5.

4 Experimental Results

We conduced an extensive experimental study, and evaluated the performance of CPHC on 19 datasets with varying characteristics. These datasets included both standard text datasets, and discretized versions of numerical datasets from the UCI machine learning dataset collection. For each dataset, we compared the classification results obtained by CPHC against existing classification algorithms. In order to ensure a fair comparison, we obtained data from the same resources, and used the same evaluation metrics as used by the existing classifiers. We do not report the details of the datasets used in our experiments here and refer the reader to [4, 5, 6].

4.1 Classification Performance

Experiments in [18, 19] indicate that *Added Value*, *Chi-Square*, *Yule's Q*, *Mutual Information*, *Certainty Factor* and *Conviction* outperform other interestingness measures [12, 27] in both global and instance-driven pattern-based hierarchical clustering contexts. Since CPHC is also based on pattern-based hierarchical clustering, we limited our experiments to these six measures. See [12, 27] for computational details of these measures.

To set values for the parameters for the CPHC algorithm in a principled way, we randomly selected a dataset, and tried a number of values for each parameter. The values that resulted in best results on the randomly selected dataset were blindly used across all datasets. Considering that text and UCI datasets are inherently different, we selected one text dataset (i.e., sports) and one UCI dataset (i.e., auto) for the parameter setting purpose. This resulted in selecting *Chi-Square* as the interestingness measure for all text datasets, and *Added Value* as the interestingness measure for all UCI datasets. In addition, we obtained *min_std_dev* = 1.5, and *maxK* = 11. Finally, we fixed *t* = 10 on all datasets (Section 3.1), and fixed *min_supp* to 2 on all small UCI datasets, and to 40 on all other datasets. Section 4.3 discusses further improvements that may be realized by tuning *min_std_dev* and *measure* for individual datasets.

Additionally, all results reported here used the 10-fold cross validation scheme (with averages of all 10 experiments reported, as usual), except on Reuters-21578 dataset, where we used the ModApte split [4] to ensure an apples-to-apples comparison with results reported by existing studies.

4.1.1 Reuters- 21578 (ModApte) text dataset. Reuters-21578 is the most-commonly used benchmark dataset to evaluate the performance of multi-class, multi-label classification algorithms. Existing studies given in [9, 30] used the precision-recall breakeven points on the ten largest categories, as the main performance criteria. We calculated these breakeven points in a way similar to [30], i.e., by changing the dominant factor, and keeping a fixed "score differentia factor" (i.e., 0.6). As mentioned above, we fixed the interestingness measure to *Chi-Square* and *min_std_dev* to 1.5.

Table 1 presents the results of this experiment. The results for Find-Sim, Naïve Bayes, Bayes-Nets, Trees (i.e., Decision-Trees), and linear-SVM are obtained from [9], while the results for ARC-BC are obtained from [1]. Note that [30] also used the same results. Finally, the results for Harmony are obtained from Table VIII of [30]. Among the ten largest categories, CPHC achieved the best break-even performance on 3 categories (i.e., crude, interest and money-fx), and ranked second on another 2 categories (i.e., acq and trade), with ranks 3-5 achieved on the remaining 4 categories. Most significantly, CPHC outperformed all existing classification algorithms in terms of micro-average performance, and also achieved a macro-average that is very close to SVM. Micro-average is calculated as the weighted (proportional to the class size) average of per-class precision-recall breakeven points, which results in an equal weight for each document, thus favoring the performance on common classes. In contrast, macro-average is obtained by first calculating the precision-recall breakpoint values for all classes, and then averaging the results. Therefore, macro-average equally weights all the classes, regardless of how many documents belong to a class.

Table 1. Breakeven performance on Reuters-21578.

Category	*Harmony*	*Find Sim*	*Naïve Bayes*	*Bayes Nets*	*Trees*	*SVM (linear)*	*ARC-BC*	*CPHC*
acq	**95.3**	64.7	87.8	88.3	89.7	93.6	90.9	94.5
corn	78.2	48.2	65.3	76.4	**91.8**	90.3	69.6	77.2
crude	85.7	70.1	79.5	79.6	85.0	88.9	77.9	**90.7**
earn	**98.1**	92.9	95.9	95.8	97.8	98.0	92.8	96.5
grain	91.8	67.5	78.8	81.4	85.0	**94.6**	68.8	91.1
interest	77.3	63.4	64.9	71.3	67.1	77.7	70.5	**81.0**
money-fx	80.5	46.7	56.6	58.8	66.2	74.5	70.5	**84.3**
ship	**86.9**	49.2	85.4	84.4	74.2	85.6	73.6	78.3
trade	**88.4**	65.1	63.9	69.0	72.5	75.9	68.0	87.9
wheat	62.8	68.9	69.7	82.7	**92.5**	91.8	84.8	83.6
micro-avg	92.0	64.6	81.5	85.0	88.4	92.0	82.1	**92.1**
macro-avg	84.5	63.7	74.8	78.8	82.2	**87.1**	76.7	86.5

4.1.2 UCI datasets. UCI machine learning datasets are also commonly used to evaluate classification algorithms. We compared the performance of CPHC against existing algorithms on 13 small and 2 large UCI datasets. To ensure fairness, we used the same pre-discretized versions of these datasets as used in [30], obtained from [6].

Table 2. Classification accuracies on 13 small UCI datasets.

	FOIL	*CPAR*	*SVM*	*Harmony*	*CPHC*
anneal	**96.90**	90.20	83.83	91.51	93.82
auto	46.10	48.00	55.50	61.00	**73.00**
breast	94.40	94.80	**96.80**	92.42	93.33
glass	49.30	48.00	46.00	49.80	**70.00**
heart	57.40	51.10	**60.36**	56.46	58.33
hepatitus	77.50	76.50	81.83	83.16	**83.33**
horsecolic	**83.50**	82.30	83.31	82.53	73.61
ionoSphere	89.50	**92.90**	89.44	92.03	92.57
iris	94.00	**94.70**	94.67	93.32	94.67
pima	73.80	**75.60**	74.18	72.34	73.16
tic-tac-toe	**96.00**	72.20	70.78	92.29	72.74
wine	86.40	92.50	**94.90**	91.94	88.24
zoo	96.00	96.00	86.00	93.00	**97.00**
average	80.06	78.06	78.28	80.91	**81.83**

Table 3. Classification accuracies on 2 large UCI datasets.

	FOIL	*CPAR*	*SVM*	*Harmony*	*CPHC*
adult	82.50	76.70	84.16	81.90	**84.95**
mushroom	99.50	98.80	99.67	99.94	**99.98**
average	91.00	87.85	91.92	90.92	**92.46**

Tables 2 and 3 present the results of this experiment. The results for FOIL, CPAR, SVM (i.e., rbf-kernel), and Harmony are obtained from tables XII and XV of [30], which also notes that C4.5, Ripper, and association-based algorithms did not perform as well on these datasets. CPHC outperformed all existing algorithms, with the highest average classification accuracies.

Table 4. Classification accuracy on the Sports dataset. SVM and Harmony used various values for *C* and minimum support. CPHC used various values for *min_supp*.

Harmony (Min support)				***SVM (C)***				***CPHC (min_supp)***			
75	100	125	150	2.0	1.0	0.5	0.25	5	10	20	30
94.2	94.9	94.3	94.1	95.79	95.79	95.76	95.72	96.40	96.24	96.12	95.98

4.1.3 Sports text dataset. We also evaluated the classification accuracy of CPHC on the Sports text dataset (i.e., TREC, original source: San Jose Mercury News). The results of SVM and Harmony are obtained from [30], which used various parameter values to tune these algorithms. We follow a similar approach and used various values for *min_supp*, which is our noise elimination parameter used in the feature selection step (Section 3.1). The values for all other parameters were kept fixed. From Table 4, we observe that CPHC resulted in better classification accuracies than both of the existing algorithms.

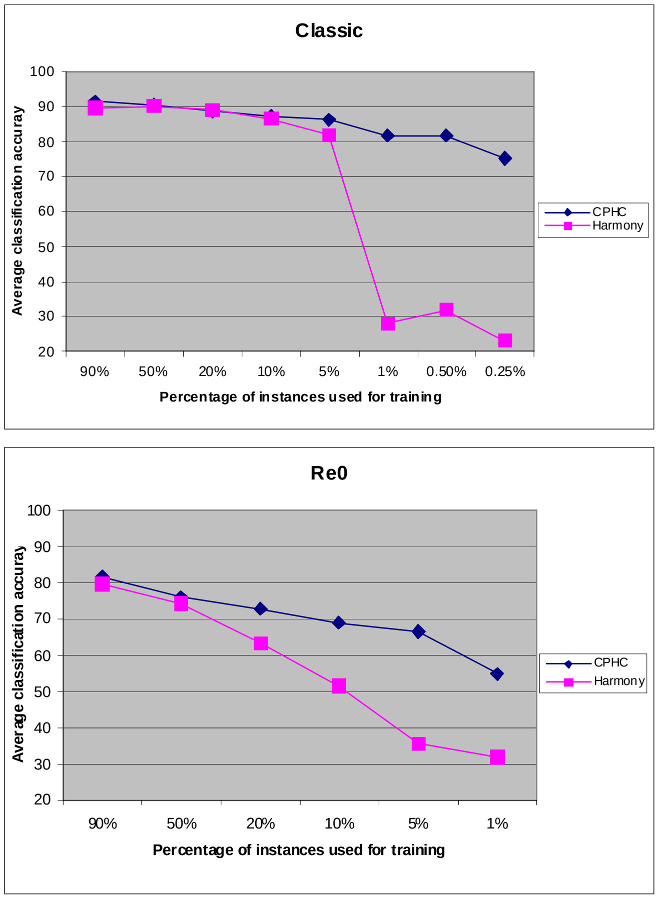

Fig. 4. Classification accuracies on Classic and Re0 datasets with increasingly sparser training data. The non-linear scale is approximately logarithmic.

4.2 Impact of the Percentage of Training Instances on Classification Performance

To evaluate how CPHC reacts to a decreasing ratio of training instances to test instances, we performed a number of experiments on the Classic and Re0 datasets obtained from [5]. In each experiment, we randomly selected *p*% of the instances as the training set (see Figure 4 for the values of *p* used on each dataset), and the rest as test set. For each value of *p*, we executed CPHC ten times and report the average classification accuracies. For comparison, we executed Harmony (i.e., executables obtained from the authors of [30]) in a similar fashion and report the average accuracies in Figure 4. Note that Harmony uses a minimum support threshold which we fixed to 1% of the training instances in each execution.

From Figure 4, we observe that the two algorithms yielded similar accuracies when a large percentage of the dataset was used as the training set. However, CPHC significantly outperformed Harmony as the size of the training set decreased. On Classic and Re0 datasets, the maximum difference in classification accuracy was as great as 53% and 31% respectively! It appears that CPHC's ability to classify "isolated" test instances, as discussed in Section 3.3, is responsible for this difference.

4.3 Optional Parameter Tuning

Table XVI of [30] presents the classification accuracies achieved on the 13 small UCI datasets by tuning SVM and Harmony using various parameter values. We follow a similar approach to demonstrate additional gains that might be realized by tuning our parameters. For this purpose, we varied *min_std_dev* between 0.9 and 2.0, in uniform intervals of 0.1, and used six different interestingness measures.

Table 5. Tuned accuracies on UCI datasets.

	Harmony	*SVM*	*CPHC*	*min_std_dev*	*Interestingness measure*
anneal	95.65	**97.26**	95.73	0.9	Certainty Factor
auto	61.50	58.90	**73.00**	1.2	Added Value
breast	**96.14**	95.09	94.06	1.3	YulesQ
glass	49.80	50.53	**75.71**	1.0	YulesQ
heart	58.40	57.46	**62.00**	1.3	Certainty Factor
hepatitis	**85.99**	85.50	84.67	1.8	Added Value
horsecolic	84.64	84.06	76.39	1.4	YulesQ
ionosphere	93.45	89.43	92.29	1.5	Added Value
iris	**95.99**	93.33	95.33	1.5	Mutual Information
pima	73.79	71.06	**75.92**	1.0	Chi Square
Tic-tac-toe	**94.09**	88.52	73.16	1.2	YulesQ
wine	94.90	**97.25**	95.88	1.0	Chi Square
zoo	96.00	97.00	**98.00**	1.1	Added Value
average	83.1	81.95	**84.01**		

Table 5 presents the best classification accuracy achieved on each dataset, along with the corresponding parameter values. For comparison sake, we also include fully tuned Harmony and SVM accuracies as reported in [30]. We observe that CPHC achieved better accuracies on 5/13 datasets, and also resulted in the highest average classification accuracy across all 13 datasets.

5 Conclusions and Future Work

The semi-supervised approach first clusters both the training and test sets together into a single cluster hierarchy, and then uses this hierarchy as a direct means for classification; this eliminates the need to train a classifier on an enhanced training set. In addition, this approach uses a novel feature selection method that ensures that all training and test instances are covered by the selected features, uses parameters that are robust across datasets with varying characteristics, and also has the positive side effect of improving the chances of classifying isolated test instances on sparse training data by inducing a form of feature transitivity. Lastly, this approach is very robust on very sparse training data.

In the future, we would like to compare CPHC against transductive learning algorithms, perform theoretical analysis to confirm our empirically observed relationship between entropy values and node pattern lengths, and extend CPHC to work in online scenarios by dynamically maintaining the cluster hierarchy as test instances arrives.

6 Acknowledgements

We would like to thank Prof. Howard Hamilton, and the anonymous reviewers for their useful comments that helped a lot in improving this paper.

References

1. Antonie, M., Zaiane, O.: Text Document Categorization by Term Association. In: Second IEEE International Conference on Data Mining (2002)
2. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: 20th International Conference on Very Large Databases, pp. 487-499 (1994)
3. Beil, F., Ester, M., Xu, X.: Frequent term-based text clustering. In: International Conference on Knowledge Discovery and Data Mining, pp. 436-442 (2002)
4. Bergsma, S.: The Reuters-21578 (ModApte) dataset. Dept. of Computer Science, University of Alberta, http://www.cs.ualberta.ca/~bergsma/HTML/Courses/650/
5. Cluto. http://glaros.dtc.umn.edu/gkhome/views/cluto
6. Coenen, F.: The LUCS-KDD Implementations of the FOIL, PRM, and CPAR algorithms. http://www.csc.liv.ac.uk/~frans/KDD/Software
7. Cohen, W.: Fast effective rule induction. In: 12th International Conference on Machine Learning (1995)
8. Dong, G., Zhang, X., Wang, L., Li, J.: CAEP: Classification by aggregating emerging patterns. Discovery Science, Volume 1721 (1999)

9. Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive Learning Algorithms and Representations for Text Categorization. In seventh international conference on Information and knowledge management, pp. 148-155 (1998)
10. Fung, B., Wang, K., Ester, M.: Hierarchical document clustering using frequent itemsets. In: SIAM International Conference on Data Mining, pp. 59-70 (2003)
11. Gabrilovich, E., Markovitch, S.: Text Categorization with Many Redundant Features: Using Aggressive Feature Selection to Make SVMs Competitive with C4.5. In: Twenty-First International Conference on Machine Learning, pp. 321-328 (2004)
12. Geng, L., Hamilton, H. J.: Interestingness Measures for Data Mining: A Survey, ACM Computing Surveys, Volume 38, No. 3 (2006)
13. Gravano, L., Ipeirotis, P., Sahami, M.: QProber: A System for Automatic Classification of Hidden-Web Databases. ACM Transactions on Information Systems, Volume 21, No. 1 (2003)
14. Kyriakopoulou, A., Kalamboukis, T.: Using clustering to enhance text classification. In: 30th annual international ACM SIGIR conference on Research and development in information retrieval (2007)
15. Li, J., Dong, G., Ramamohanarao, K., and Wong, L., DeEPs: A New Instance based Discovery and Classification System. Machine Learning, Volume 54, No. 2 (2004)
16. Li, W., Han, J., Pei, J.: CMAR: Accurate and Efficient Classification based on multiple class-association rules. In: First IEEE International Conference on Data Mining (2001)
17. Liu, B., Hsu, W., Ma, Y.: Integrating Classification and Association Rule Mining. In: Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1998)
18. Malik, H. H., Kender, J. R.: High Quality, Efficient Hierarchical Document Clustering Using Closed Interesting Itemsets. In: Sixth IEEE International Conference on Data Mining, pp. 991-996 (2006)
19. Malik, H. H., Kender, J. R.: Instance Driven Hierarchical Clustering of Document Collections. In: From Local Patterns to Global Models Workshop, European Conference on Machine Learning and Practice of Knowledge Discovery in Databases (2008)
20. Pereira, F., Tishby, N., Lee, L.: Distributional clustering of English words. In: 31st Annual Meeting of the Association for Computational Linguistics (1993)
21. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufman, ISBN:1-55860-238-0 (1993)
22. Quinlan, J., Cameron-Jones, R.: FOIL: A Midterm Report. In: European Conference on Machine Learning (1993)
23. Raskutti, B., Ferr, H., Kowalczyk, A.: Using unlabeled data for text classification through addition of cluster parameters. In: 9th International Conference on Machine Learning (2002)
24. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys, Volume 34, No. 1 (2002)
25. Slonim, N., Tishby, N.: The power of word clustering for text classification. In: European Colloquium on IR Research (2001)
26. Turns out some dinosaurs could swim, http://www.cnn.com
27. Tan, P., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: 8th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 32–41 (2002)
28. Vapnik, V. N.: Statistical Learning Theory. Wiley, ISBN: 0-47-103003-1 (1998)
29. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2003)
30. Wang, J., Karypis, G.: On Mining Instance-Centric Classification Rules. IEEE Transactions on Knowledge and Data Engineering, Volume 18, No. 11 (2006)
31. Xiong, H., Steinbach, M., Tan, P.-N., Kumar, V.: HICAP: Hierarchical Clustering with Pattern Preservation. In: SIAM International Conference on Data Mining (2004)

32. Yin, X., Han, J.: CPAR: Classification based on Predictive Association Rules. In: SIAM International Conference on Data Mining (2003)
33. Yu, H., Searsmith, D., Li, X., Han, J.: Scalable Construction of Topic Directory with Nonparametric Closed Termset Mining. In: Fourth IEEE International Conference on Data Mining, pp. 563-566 (2004)
34. Zeng, H. J., Wang, X.H., Chen, Z., Lu, H., Ma, W. Y.: CBC: Clustering based text classification requiring minimal labeled data. In: Third IEEE International Conference on Data Mining (2003)
35. Zhao, Y., Karypis, G.: Hierarchical Clustering Algorithms for Document Datasets. Data Mining and Knowledge Discovery, Volume 10, pp. 141--168, No. 2 (2005)