

Generalization and Decision Tree Induction: Efficient Classification in Data Mining

Micheline Kamber Lara Winstone Wan Gong Shan Cheng Jiawei Han
Database Systems Research Laboratory
School of Computing Science
Simon Fraser University, B.C., Canada V5A 1S6
{kamber, winstone, wgong, shanc, han}@cs.sfu.ca

Abstract

Efficiency and scalability are fundamental issues concerning data mining in large databases. Although classification has been studied extensively, few of the known methods take serious consideration of efficient induction in large databases and the analysis of data at multiple abstraction levels. This paper addresses the efficiency and scalability issues by proposing a data classification method which integrates attribute-oriented induction, relevance analysis, and the induction of decision trees. Such an integration leads to efficient, high-quality, multiple-level classification of large amounts of data, the relaxation of the requirement of perfect training sets, and the elegant handling of continuous and noisy data.

1. Introduction

Computational efficiency and scalability are two important and challenging issues in data mining. Data mining is the automated discovery of non-trivial, previously unknown, and potentially useful patterns embedded in databases [9]. The increasing computerization of all aspects of life has led to the storage of massive amounts of data. Large scale data mining applications involving complex decision-making can access billions of bytes of data. Hence, the efficiency of such applications is paramount.

Classification is a key data mining technique whereby database tuples, acting as *training samples*, are analyzed in order to produce a model of the given data [5, 8, 23]. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the *classifying attribute*. Once derived, the classification model can be used to categorize future data samples, as well as provide a better understanding of the database contents. Classification has numerous applications including credit approval, product marketing, and medical diagnosis.

A number of classification techniques from the statistics and machine learning communities have been proposed [7,

25, 26, 30]. A well-accepted method of classification is the induction of decision trees [3, 25]. A decision tree is a flow-chart-like structure consisting of internal nodes, leaf nodes, and branches. Each internal node represents a decision, or test, on a data attribute, and each outgoing branch corresponds to a possible outcome of the test. Each leaf node represents a class. In order to classify an unlabeled data sample, the classifier tests the attribute values of the sample against the decision tree. A path is traced from the root to a leaf node which holds the class predication for that sample. Decision trees can easily be converted into IF-THEN rules [26] and used for decision-making.

The efficiency of existing decision tree algorithms, such as ID3 [25] and CART [3], has been well established for relatively small data sets [16, 22, 29]. Efficiency and scalability become issues of concern when these algorithms are applied to the mining of very large, real-world databases. Most decision tree algorithms have the restriction that the training tuples should reside in main memory. In data mining applications, very large training sets of millions of examples are common. Hence, this restriction limits the scalability of such algorithms, where the decision tree construction can become inefficient due to swapping of the training samples in and out of main and cache memories.

The induction of decision trees from very large training sets has been previously addressed by the SLIQ [20] and SPRINT [27] decision tree algorithms. These propose pre-sorting techniques on disk-resident data sets that are too large to fit in memory. While SLIQ's scalability, however, is limited by the use of a memory-resident data structure, SPRINT removes all memory restrictions and hence can handle data sets that are too large for SLIQ [27]. Unlike SLIQ and SPRINT, which operate on the raw, low-level data, we address the efficiency and scalability issues by proposing a different approach, consisting of three steps: 1) *attribute-oriented induction* [11, 13], where concept hierarchies are used to generalize low-level data to higher level concepts, 2) *relevance analysis* [10], and 3) *multi-level mining*, whereby decision trees can be induced at different levels of abstraction.

Attribute-oriented induction, a knowledge discovery tool which allows the generalization of data, offers two major advantages for the mining of large databases. First, it allows the raw data to be handled at higher conceptual levels. Generalization is performed with the use of attribute concept hierarchies, where the leaves of a given attribute's concept hierarchy correspond to the attribute's values in the data (referred to as primitive level data) [13]. Generalization of the training data is achieved by replacing primitive level data (such as numerical values for a *GPA*, or *grade point average* attribute) by higher level concepts (such as ranges $3.6 - 3.8$, $3.8 - 4.0$, or categories *good* or *excellent*). Hence, attribute-oriented induction allows the user to view the data at more meaningful abstractions. In many decision tree induction examples, such as in [25], the data to be classified are presented at a relatively high concept level, such as "mild" (temperature) and "high" (humidity). This implies that some preprocessing may have been performed on the primitive data to bring it to a higher concept level. Attribute-oriented induction is useful in that it allows the generalization of data to *multiple-level* generalized concepts. This is particularly useful for continuous-valued attributes.

Furthermore, attribute-oriented induction addresses the scalability issue by compressing the training data. The generalized training data will be much more compact than the original training set, and hence, will involve fewer input/output operations. As we will illustrate in section 4, the efficiency of the resulting decision tree induction will be greatly improved.

The second step of our approach aids scalability by performing attribute relevance analysis [10] on the generalized data, prior to decision tree induction. This process identifies attributes that are either irrelevant or redundant. Including such attributes in the generalized data would slow down, and possibly confuse, the classification process. This step improves classification efficiency and enhances the scalability of classification procedures by eliminating useless information, thereby reducing the amount of data that is input to the classification stage.

The third and final step in our approach is multi-level mining. We propose two algorithms which allow the induction of decision trees at different levels of abstraction by employing the knowledge stored in the concept hierarchies. Furthermore, once a decision tree has been derived, the concept hierarchies can be used to generalize or specialize individual nodes in the tree, allowing attribute rolling-up or drilling-down, and reclassification of the data for the newly specified abstraction level. This interactive feature executes in our *DBMiner* data mining system [14] with fast response.

This paper is organized as follows. Section 2 describes the attribute-oriented induction, relevance analysis, and multi-level mining components of our proposed approach.

Each component has been implemented in *DBMiner* [14]. Section 3 presents *MedGen* and *MedGenAdjust*, our proposed multi-level decision tree induction algorithms. A performance evaluation is given in Section 4. Section 5 addresses related issues, such as classification accuracy. Conclusions and future work are discussed in Section 6.

2. Proposed approach: generalization-based induction of decision trees

We address efficiency and scalability issues regarding the data mining of large databases by proposing a technique composed of the following three steps: 1) generalization by attribute-oriented induction, to compress the training data. This includes storage of the generalized data in a multi-dimensional data cube to allow fast accessing, 2) relevance analysis, to remove irrelevant data attributes, thereby further compacting the training data, and 3) multi-level mining, which combines the induction of decision trees with knowledge in concept hierarchies. This section describes each step in detail.

2.1. Attribute-oriented induction

In real-world applications, data mining tasks such as classification are applied to training data consisting of thousands or millions of tuples. Applying attribute-oriented induction prior to classification substantially reduces the computational complexity of this data-intensive process [14].

Data are compressed with a *concept tree ascension* technique which replaces attribute values by generalized concepts from corresponding attribute concept hierarchies [11]. Each generalized tuple has a *count* associated with it. This registers the number of tuples in the original training data that the generalized tuple now represents. The *count* information represents statistical data and is used later in the classification process, and enables the handling of noisy and exceptional data.

Concept hierarchies may be provided by domain experts or database administrators, or may be defined using the database schema [11]. Concept hierarchies for numeric attributes can be generated automatically [12]. In addition to allowing the substantial reduction in size of the training set, concept hierarchies allow the representation of data in the user's vocabulary. Hence, aside from increasing efficiency, attribute-oriented induction may result in classification trees that are more understandable, smaller, and therefore easier to interpret than trees obtained from methods operating on ungeneralized (larger) sets of low-level data (such as [20, 27]). The degree of generalization is controlled by an empirically-set *generalization threshold*. If the number of distinct values of an attribute is less than or equal to this threshold, then further generalization of the attribute is halted.

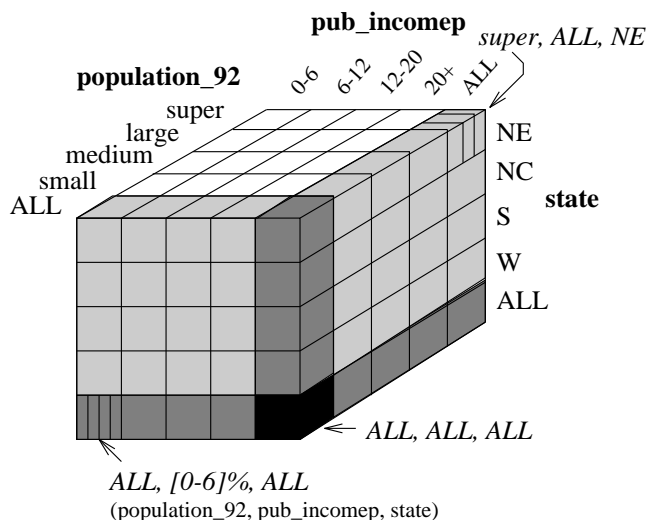


Figure 1. A multi-dimensional data cube.

Attribute-oriented induction also performs generalization by *attribute removal* [11]. In this technique, an attribute having a large number of distinct values is removed if there is no higher level concept for it. Attribute removal further compacts the training data and reduces the bushiness of resulting trees.

In our approach, the generalized data are stored in a *multi-dimensional data cube* [14], similar to that used in data warehousing [15]. This is a multi-dimensional array structure. Each dimension represents a generalized attribute, and each cell stores the value of some aggregate attribute, such as the *count* information described earlier. Consider a *CITYDATA* database consisting of statistics describing incomes and populations, collected for cities and counties in the United States. Attributes in the database include *state*, *population_92* (city population in 1992), and *pub_incomep* (percentage of population receiving public assistance). A multi-dimensional data cube, *citydata*, is depicted in Figure 1. The original values of *state* have been generalized to *NE* (North_East), *NC* (North_Central), *S* (South), and *W* (West), while *population_92* has been generalized to *small_city*, *medium_city*, *large_city* and *super_city*, and *pub_incomep* has been generalized to *0-6%*, *6-12%*, *12-20%*, and *20+%*. Each dimension includes a value *ALL*, representing the aggregate sum of the entire dimension. The advantage of the multi-dimensional structure is that it allows fast indexing to cells (or slices) of the cube. For instance, one may easily and quickly access the total count of cities for which *0-6%* of the 1992 population received public assistance, for all states and population sizes, combined (i.e., cell $\langle ALL, 0-6\%, ALL \rangle$).

We illustrate the ideas of attribute-oriented induction with

$$\begin{aligned} & \{ USA \} \subset \{ ANY \} \\ & \{ North_East, North_Central, South, West \} \subset \{ USA \} \\ & \{ New_England, Middle_Atlantic \} \subset \{ North_East \} \\ & \{ Mountain, Pacific_East, Pacific_West \} \subset \{ West \} \\ & \dots \{ Alaska, Hawaii \} \subset \{ Pacific_West \} \\ & \{ small_city, medium_city, large_city, super_city, other \} \subset \\ & \quad \{ ANY \} \\ & \{ 10000 \sim 20000, 20000 \sim 30000, 30000 \sim 40000, \\ & \quad 40000 \sim 50000 \} \subset \{ small_city \} \\ & \dots \{ 1000000 \sim 9999999 \} \subset \{ super_city \} \end{aligned}$$

Figure 2. Concept hierarchies for attributes *state* and *population_92* of the *CITYDATA* database.

an example using the described database. The data mining task is to classify the data according to the database attribute *median_income* (median family income), based on the attributes *city_area* (in square miles), *state*, *population_92*, and *pub_incomep*. This task is specified in our data mining query language, DMQL [14], as shown below in Example 2.1.

Example 2.1: *Classification task.*

```
classify CITYDATA
according to median_income
in relevance to city_area, state,
           population_92, pub_incomep
from INCOME, POPULATION
where INCOME.place = POPULATION.place
and INCOME.state = POPULATION.state
```

Prior to applying attribute-oriented induction, the above query retrieves the task-relevant data by performing a relational query on *CITYDATA*. Tuples not satisfying the *where* clause are ignored, and only the data concerning the relevant attributes specified in the *in relevance to* clause, and the class attribute, *median_income*, are collected. An example of such task-relevant training data is shown in Table 1.

| city_area | state | population_92 | pub_incomep | median_income |
|-----------|-----------|---------------|-------------|---------------|
| 20.2 | Alabama | 27115 | 10.8 | 26067.0 |
| ... | ... | ... | ... | ... |
| 11.3 | Wisconsin | 62149 | 5.3 | 36085.0 |
| 20.6 | Wyoming | 47800 | 5.9 | 33366.0 |

Table 1. Task-relevant data from the *CITYDATA* database (ungeneralized).

Attribute-oriented induction is performed on the set of relevant data. An intermediate generalized relation,

achieved by concept ascension using the concept hierarchies of Figure 2 for the attributes *state* and *population_92*, is displayed in Table 2. Owing to the attribute removal technique, *city_area* does not appear in the resulting table because it has many distinct values and no concept hierarchy with which it could be further generalized. Identical tuples for Table 1 were merged while collecting the count (*cnt*) information. In practice, the resultant table will be substantially smaller than the original database, as well as the ungeneralized task-relevant data.

| state | popula- tion_92 | pub_ incomep | median_ income | cnt |
|------------|--------------------|-----------------|-------------------|-----|
| North_East | large_city | 0%-6% | 45Ks- | 1 |
| North_East | small_city | 0%-6% | 30Ks-45Ks | 14 |
| ... | ... | ... | ... | ... |
| West | large_city | 6%-12% | 30Ks-45Ks | 27 |

Table 2. Task-relevant data, generalized to an intermediate concept level, with count (*cnt*).

How high a level should each attribute be generalized prior to the application of relevance analysis and multi-level mining? There are at least three possibilities. First, generalization may proceed to the *minimally generalized concept level*, based on the attribute generalization thresholds described above. A relation is minimally generalized if each attribute satisfies its generalization threshold, and if specialization of any attribute would cause the attribute to exceed the generalization threshold [11]. A disadvantage of this approach is that, if each database attribute has many discrete values, the decision tree induced will likely be quite bushy and large.

On the other extreme, generalization may proceed to very high concept levels. Since the resulting relation will be rather small, subsequent decision tree induction will be more efficient than that obtained from minimally generalized data. However, by overgeneralizing, the classification process may lose the ability to distinguish interesting classes or subclasses, and thus may not be able to construct meaningful decision trees.

A trade-off, which we adopt, is to generalize to an intermediate concept level. Such a level can be specified either by a domain expert, or by a threshold which defines the desired number of distinct values for each attribute (such as “4-8 distinct attribute values”). For example, one may generalize the attribute values for *state* to $\{North_East, North_Central, South, West\}$. It is possible, however, that an intermediate level derived from a predefined concept hierarchy may not best suit the given classification task. In such cases, a level-adjustment process should be introduced to im-

prove the classification quality. In section 2.3, we describe a technique for level-adjustment that is integrated with the induction of decision trees.

2.2. Relevance analysis

In the second step of our approach, we apply attribute relevance analysis to the generalized data obtained from attribute-oriented induction. This further reduces the size of the training data.

A number of statistical and machine learning-based techniques for relevance analysis have been proposed in the literature [2, 10, 17]. We employ an information-theoretic asymmetric measure of relevance known as the *uncertainty coefficient* [19]. This coefficient is based on the information gain attribute selection measure used in C4.5 [26] for building decision trees.

The relevance analysis is performed as follows. Let the generalized data be a set P of p data samples. Suppose the classifying attribute has m distinct values defining m distinct classes P_i (for $i = 1, \dots, m$). Suppose P contains p_i samples for each P_i , then an arbitrary sample belongs to class P_i with probability p_i/p . The expected information needed to classify a given sample is given by,

$$I(p_1, p_2, \dots, p_m) = - \sum_{i=1}^m \frac{p_i}{p} \log_2 \frac{p_i}{p}. \quad (2.1)$$

An attribute A with values $\{a_1, a_2, \dots, a_k\}$ can be used to partition P into $\{C_1, C_2, \dots, C_k\}$, where C_j contains those samples in C that have value a_j of A . Let C_j contain p_{ij} samples of class P_i . The expected information based on the partitioning by A is given by,

$$E(A) = \sum_{j=1}^k \frac{p_{1j} + \dots + p_{mj}}{p} I(p_{1j}, \dots, p_{mj}). \quad (2.2)$$

Thus, the information gained by branching on A is,

$$gain(A) = I(p_1, p_2, \dots, p_m) - E(A). \quad (2.3)$$

The uncertainty coefficient for attribute A , i.e., $U(A)$, is obtained by normalizing the information gain of A so that $U(A)$ ranges from 0 (meaning statistical independence between A and the classifying attribute) to 1 (strongest degree of relevance between the two attributes),

$$U(A) = \frac{I(p_1, p_2, \dots, p_m) - E(A)}{I(p_1, p_2, \dots, p_m)}. \quad (2.4)$$

The user has the option of retaining either the n most relevant attributes, or all attributes whose uncertainty coefficient value is greater than a pre-specified uncertainty threshold, where n and the threshold are user-defined. Note that it is

much more efficient to apply the relevance analysis to the generalized data rather than to the original training data. Such analysis on large sets of (ungeneralized) data can be computationally expensive. Hence, by applying attribute-oriented induction in step one in order to reduce the size of the training data, we reduce the amount of computation required in step two of our approach. Furthermore, relevance analysis contributes to step 3 by removing attributes that would otherwise slow down and possibly confuse the classification process.

2.3. Multi-level mining

The third and final step of our method is multi-level mining. This combines decision tree induction of the generalized data obtained in steps 1 and 2 (attribute-oriented induction and relevance analysis) with knowledge in the concept hierarchies. In this section, we introduce two algorithms for multi-level mining. A detailed description of each algorithm follows in Section 3.

In section 2.1, the degree to which attribute-oriented induction should be applied in order to generalize data was discussed. Generalization to a minimally generalized concept level can result in very large and bushy trees. Generalization to very high concept levels can result in decision trees of little use since overgeneralization may cause the loss of interesting and important subconcepts. It was determined that it is most desirable to generalize to some *intermediate* concept level, set by a domain expert or controlled by a user-specified threshold.

We also noted that generalization to an intermediate level derived from a predefined concept hierarchy may not best suit the given classification task. Thus to improve classification quality, it can be useful to incorporate a **concept level-adjustment** procedure into the process of inducing decision trees. Hence, we propose two algorithms: **MedGen** and **MedGenAdjust**. Each algorithm operates on training data that have been generalized to an intermediate level by attribute-oriented induction, and for which unimportant attributes have been removed by relevance analysis. **MedGen** simply applies decision tree induction to the resulting data. **MedGenAdjust** applies a modified decision tree algorithm which allows the adjustment of concept levels for each attribute in an effort to enhance classification quality. Since both algorithms confine their processing to relatively small generalized relations, rather than repeatedly accessing the (initially large) raw data, both algorithms should have a reasonable processing cost.

3. Proposed algorithms: MedGen and MedGenAdjust

This section describes the **MedGen** and **MedGenAdjust** algorithms for classification of large databases in greater detail. Both algorithms apply attribute-oriented induction to generalize data to an intermediate level, followed by relevance analysis, as means of compressing the original large training set. Both **MedGen** and **MedGenAdjust** induce decision tree classifiers from the generalized relevant data. While **MedGen** directly applies a decision tree algorithm to the generalized data, **MedGenAdjust** allows for dynamic adjustment between different levels of abstraction during the tree building process.

The decision tree induction algorithm on which **MedGen** and **MedGenAdjust** are based is C4.5 [26] (an earlier version of which is known as ID3 [25]). C4.5 was chosen because it is generally accepted as a standard for decision tree algorithms, and has been extensively tested. It has been used in many application areas ranging from medicine [18] to game theory [24], and is the basis of several commercial rule-induction systems [25]. Furthermore, C4.5 allows the use of an attribute selection measure known as *information gain* [25]. In a comparative study of selection measures, information gain was found to produce accurate and small trees [4]. C4.5 has been repeatedly shown to perform well on relatively small data sets [16, 22, 29]. Section 3.1 outlines the C4.5 algorithm. **MedGen** and **MedGenAdjust** are described in Sections 3.2 and 3.3, respectively.

3.1. C4.5 overview

The C4.5 method [25, 26] is a greedy tree growing algorithm which constructs decision trees in a top-down recursive divide-and-conquer strategy. The tree starts as a single node containing the training samples. If the samples are all of the same class, then the node becomes a leaf and is labeled with that class. Otherwise, the algorithm uses equations 2.1 – 2.3 to select the attribute with the highest information gain. This attribute becomes the “decision” or “test” attribute at the node. A branch is created for each value of the decision attribute, and the samples are partitioned accordingly. The algorithm uses the same process recursively to form a decision tree for each partition. The recursive partitioning stops only when all samples at a given node belong to the same class, or when there are no remaining attributes on which the samples may be further partitioned.

3.2. The MedGen algorithm

The **MedGen** algorithm integrates attribute-oriented induction and relevance analysis with a slightly modified version of the C4.5 decision tree algorithm [26].

MedGen introduces two thresholds which are not part of C4.5. These are an *exception threshold* and a *classification threshold*. A criticism of C4.5 is that, because of the recursive partitioning, some resulting data subsets may become so small that partitioning them further would have no statistically significant basis. The maximum size of such "insignificant" data subsets can be statistically determined. To deal with this problem, MedGen introduces an exception threshold. If the portion of samples in a given subset is less than the threshold, further partitioning of the subset is halted. Instead, a leaf node is created which stores the subset and class distribution of the subset samples.

Moreover, owing to the large amount, and wide diversity, of data in large databases, it may not be reasonable to assume that each leaf node will contain samples belonging to a common class. MedGen addresses this problem by employing a classification threshold, κ . Further partitioning of the data subset at a given node is terminated if the percentage of samples belonging to any given class at that node exceeds the classification threshold. Our classification threshold is similar to the precision threshold introduced by [1]. Use of such thresholds for dynamic pruning may be more efficient than traditional tree pruning strategies [21] which first fully grow a tree and then prune it. The MedGen algorithm is outlined below.

Algorithm 3.1 (MedGen) *Perform data classification using a prespecified classifying attribute on a relational database by integration of attribute-oriented induction and relevance analysis with a decision tree induction method.*

Input. 1) a data classification task which specifies the set of relevant data and a classifying attribute in a relational database DB , 2) a set of concept hierarchies on attributes A_i , 3) τ_i , a set of *attribute (generalization) thresholds* for attributes A_i , 4) an exception threshold ϵ , and 5) a classification threshold κ .

Output. A classification of the set of data and a set of classification rules.

Method.

1. **Data collection:** Collect the relevant set of data, R_0 , by relational query processing.
2. **Attribute-oriented induction:** Using the concept hierarchies, perform attribute-oriented induction to generalize R_0 to an intermediate level. The resultant relation is R_1 . The generalization level is controlled by τ_i , a set of *attribute thresholds* associated with attributes A_i .
3. **Relevance analysis:** Perform relevance analysis on the generalized data relation, R_1 . The information theoretic-based uncertainty coefficient [19] is used for this purpose, although other methods, such as those suggested in [2, 17] could be used. The resulting data relation is R_2 .

4. **Decision-tree generation**

- (a) Given the generalized and relevant data relation, R_2 , compute the information gain for each candidate attribute using the equations (2.1) – (2.3). Select the candidate attribute which gives the maximum information gain as the decision or "test" attribute at this current level, and partition the current set of objects accordingly. Although information gain is used here, as in C4.5 [25], alternatively other selection measurements (such as the gini-index [20, 30]) could be used.
- (b) For each subset created by the partitioning, repeat Step 4a to further classify data until either (1) all or a substantial proportion (no less than κ , the classification threshold) of the objects are in one class, (2) no more attributes can be used for further classification, or (3) the percentage of objects in the subclass (with respect to the total number of training samples) is below ϵ , the exception threshold.

5. **Classification-rule generation:** Generate rules according to the decision tree so derived (as in [26]). □

Rationale of Algorithm 3.1. Step 1 is relational query processing. Step 2 has been verified in [11, 13]. Step 3 eliminates irrelevant attributes from the generalized data. Step 4 is essentially the C4.5 algorithm whose correctness has been shown in [25, 26]. Our modification uses quantitative information collected in Step 1 to terminate classification if the frequency of the majority class in a given subset is greater than the classification threshold, or if the percentage of training objects represented by the subset is less than the exception threshold. Otherwise, further classification will be performed recursively. Step 5 generates rules according to the decision tree which reflect the general regularity of the data in the database. □

3.3. **The MedGenAdjust algorithm**

The MedGenAdjust algorithm performs the same steps as the MedGen algorithm, yet it replaces the **decision-tree generation step** (Step 4) by **level-adjusted decision-tree generation**, an integration of the decision tree generation process with abstraction-level adjustment. This allows the generalization and/or specialization of the abstraction level of individual decision nodes in the tree.

The **level-adjustment** process consists of *node-merge*, *node-split*, and *split-node-merge* operations. We say that a node *largely* belongs to a given class if the number of class samples it contains exceeds the classification threshold, κ . In *node-merge*, if some child nodes at a selected concept level share the same parent, and largely belong to the same

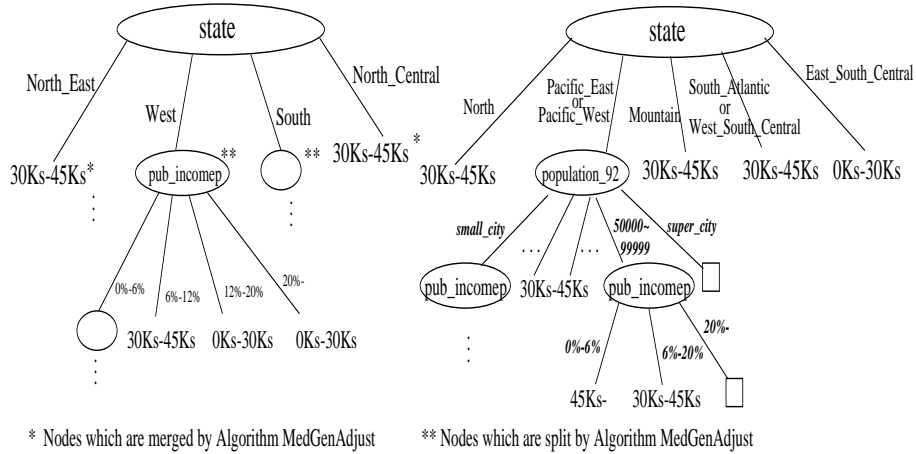


Figure 3. Decision trees obtained for *median_income* from MedGen (left), and MedGenAdjust (right).

class, then these nodes can be merged into one. A set of nodes representing numeric ranges should be merged together if such a merge would result in a range extension. In *node-split*, if a node does not largely belong to a class, but some of its sub-nodes do, then the node can be split based on the grouping of its sub-nodes. In *split-node-merge*, some split nodes may share the same parent and largely belong to the same class, so these nodes can be merged together. Moreover, certain split nodes may be further merged with neighboring nodes. Node-merging can reduce the number of disjunctions in the derived classification rules, simplifying the classification results.

For example, consider the classification task of Example 2.1, as well as the generalized *CITYDATA* of Table 2, and the concept hierarchies of Figure 2. To classify data according to *median_income*, the attribute *state* can be examined at an intermediate level, consisting of values $\{North_East, North_Central, South, West\}$. If *North* is made up of *North_Central* and *North_East*, and most cities in these regions have a high median income, then *North_Central* and *North_East* can be merged into a single node, designated as *North* (Figure 3). On the other hand, if cities in the *West* show no obvious median income class, then this category may be split. Recall from Figure 2 that *West* consists of three subregions, namely $\{Mountain, Pacific_East, Pacific_West\}$. If the *Mountain* subregion shows an obvious class of 30Ks - 45Ks for *median_income*, while the other two subregions show no obvious class, then the node *West* should be split into two sub-nodes, *Mountain* and *Pacific_East_or_Pacific_West*. The *South* node may be split in a similar fashion. Figure 3 shows the resulting tree.

The level-adjustment process can be performed on each candidate attribute. The level-adjusted attribute with the largest information gain is selected as a decision node and is used to partition the training data into subsets. The algorithm

is recursively applied to each subset, using the same stopping condition as for MedGen. The level-adjustment procedure (Step 4 of MedGenAdjust) is outlined below.

Step 4. Level-adjusted decision-tree generation.

1. For each attribute A_i , do $level_adjustment(A_i)$.

The level-adjustment procedure finds a partitioning for each A_i by performing *node-merge* (generalization), *node-split* (specialization) and *node-split-merge* (generalization and specialization), and can be performed on all relevant attributes in parallel.

2. Compute the information gain for each candidate attribute's adjusted partitioning (obtained in 1). Select one candidate attribute as the decision attribute based on the calculated information gains obtained.
3. Recursively perform Step 4 for each subclass (node) created by the partitioning until either (1) all or a substantial proportion (no less than κ) of the objects are in one class, (2) no more attributes can be used for further classification, or (3) the size of the node falls below the specified exception threshold ϵ .

The result of level-adjustment may be a partition that occurs at multiple levels of abstraction, with the principal motivating factor being improved classification through dynamic adjustment.

4. Performance Evaluation

This section describes experiments conducted to study the efficiency and scalability of our proposed approach to the classification of large databases. Recall that prior to decision tree induction, the task relevant data can be generalized to a number of different degrees, e.g.,

1. No generalization: The training data are not generalized at all. This is the case for typical applications of decision tree classifiers, such as C4.5 [26]. We call such an algorithm NoGen.
2. Minimal-level generalization: The training data are generalized to a minimally generalized concept level (see Section 2.1). Subsequently, decision tree induction is applied. We call such an algorithm MinGen.
3. Intermediate-level generalization: The training data are generalized to an intermediate concept level before decision tree induction. This is the case for the proposed MedGen and MedGenAdjust algorithms.
4. High-level generalization: The training data are generalized to a high concept level before decision tree induction. We call such an algorithm HiGen.

Since the generalized data induced in HiGen will typically be much smaller than that in the other algorithms listed here, we would expect it to be the most efficient. However, as discussed earlier, we do not expect such classifications to be meaningful due to overgeneralization. Hence, we do not include it in our study. Furthermore, we expect NoGen to be the least efficient of all methods. The NoGen approach involves working on a very large set of raw data, and thus entails heavy data accessing. In contrast, classification on generalized data will be faster since it will be working with a smaller generalized relation. Our strategy of storing data in a multi-dimensional data cube results in increased computation efficiency. Granted, some overhead is required to set up the cube. However, the subsequent advantage of stored data in a data cube is that it allows fast indexing to cells (or slices) of the cube. Recall that C4.5 requires the calculation of information gain for each attribute, considering each attribute value, at each node [26]. This results in heavy computation from the calculation of memory addresses and subsequent accessing of the raw data. Fast accessing and quick calculation of addresses are a feature of the data cube's storage and indexing system [14, 15]. Hence, our experiments focus on comparing the efficiency of MinGen, MedGen, and MedGenAdjust. Classification accuracy is discussed in Section 5.

All experiments were conducted on a Pentium-166 PC with 64 MB RAM, running Windows/NT. The algorithms were implemented in DBMiner [14] using Microsoft Visual C++. (DBMiner is accessible on the web at <http://db.cs.sfu.ca/DBMiner>). The results were obtained from the execution of a query from the *CITYDATA* database. The query was executed ten times, and the average CPU time in milliseconds was calculated. The database consists of over 29 MB, and was described in Section 2.1. Figure 4 graphs the execution time, in CPU milliseconds, for each algorithm with various classification threshold settings. As expected, the execution time for each algorithm generally increases with the classification threshold. The algorithms

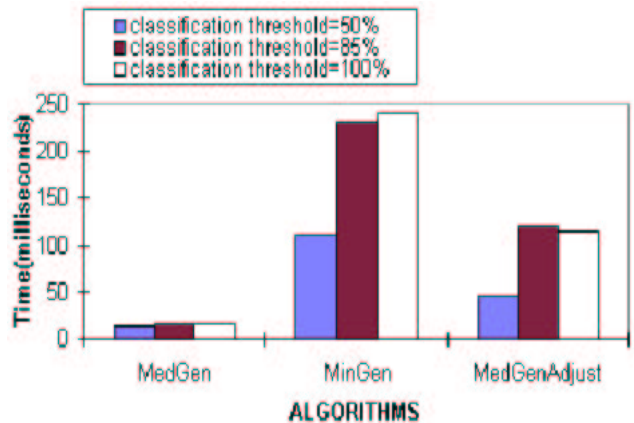


Figure 4. Average execution time (in CPU milliseconds) per algorithm for various classification threshold settings.

were tested with the exception threshold set at 0.5%. All threshold values were empirically-determined. There were 1071 tuples in the task-relevant, ungeneralized data relation, 855 tuples in the minimal concept level generalized relation (for MinGen), and 97 tuples in the intermediate level generalized relation (processed by MedGen and MedGenAdjust).

In order to generalize the data to an intermediate concept level, MedGen and MedGenAdjust must spend more time than MinGen on attribute-oriented induction. Yet, because the generalized intermediate level data are much smaller than the generalized minimal level data, both algorithms require much less time to induce decision trees than does MinGen. Hence, MinGen is the least efficient of the three algorithms. Thus, we see that generalizing training data to an intermediate level does improve the efficiency of decision tree induction. As expected, MedGen is more efficient than MedGenAdjust due to the extra processing time incurred by MedGenAdjust for level-adjustment. However, with MedGenAdjust, a balance is struck between user-desired levels and automatic refinement of the abstraction level to ensure classification quality. On average, trees induced by MinGen, MedGen, and MedGenAdjust had 72, 36, and 37 leaves, respectively. Thus, generalization also reduces the average size of the induced decision trees, thereby contributing to improved understandability of the classification results. MedGen with dynamic pruning (using an empirically set classification threshold of 85% to control the growth of branches during construction of the tree) was compared with MedGen with post-pruning (employing a classification threshold of 100% to avoid pre-pruning). The error-cost complexity post-pruning algorithm [3] was used since it has

been shown to produce small and accurate decision trees [21]. Although post-pruning requires greater computational effort since it must grow the tree fully, and then generate and examine variations of the tree pruned to different degrees in order to find the most accurate tree, we found no statistically significant difference in the accuracy of the decision trees obtained from the dynamic and post-pruning strategies.

Finally, the user can perform additional multi-level mining on a displayed tree by clicking on any nonleaf node to expand it (showing all of the node's children) or shrink it (collapsing all of the node's descendants into the node itself). Storage of the generalized data in the multi-dimensional data cube allows for fast attribute rolling-up or drilling-down, and reclassification of the data for the newly specified abstraction level. This interactive feature of DBMiner is fast, executing in real time, and allows the user to gain additional insight into the classified data.

5. Discussion

Classification accuracy. In classification problems, it is commonly assumed that all objects are uniquely classifiable, i.e., that each training sample can belong to only one class. In addition to efficiency, classification algorithms can then be compared according to their accuracy. Accuracy is measured using a *test* set of objects for which the class labels are known. Accuracy is estimated as the number of correct class predictions, divided by the total number of test samples. To classify an object, a path is traced (based on the attribute values of the object) from the root of the tree to a leaf node. The majority class at that node is deemed the class prediction of the given object. For each test object, the class prediction is compared to the known class of the object. If the two match, the class prediction is counted as correct. Otherwise, it is counted as incorrect.

However, owing to the wide diversity of data in large databases, it is not reasonable to assume that all objects are uniquely classifiable. Rather, it is more probable to assume that each object may belong to more than one class. How then, can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, since it does not take into account the possibility of samples belonging to more than one class. This problem has not yet been adequately solved in the literature.

A *second guess* heuristic has been used to address this situation [28] whereby a classification prediction is counted as correct if it agrees with the majority class of the leaf at which the object "falls", or if it agrees with the second class in majority at that leaf. Class predictions that disagree with the first two classes in majority at the leaf, are counted as incorrect. Although this method does take into consideration, in some degree, the non-unique classification of objects, it is not a complete solution.

It is difficult to compare the accuracy of decision trees that are based on the same classifying attribute, yet that use different levels of abstraction. For example, given *population_92* as the classifying attribute, one tree may use the values from, say, a low level of the attribute's concept hierarchy as class values (e.g., 10000~20000, . . . , 1000000~9999999), while another may use the values from a higher level (e.g., *small_city*, . . . , *super_city*). This situation can make it difficult to compare decision trees obtained from MedGen and MedGenAdjust. In these cases, the user plays an important role in judging the classification quality.

As we have seen, it is difficult to assess the accuracy of classifiers on large databases. Presently, we are investigating some form of *weighted* accuracy measure which considers the correctness of objects belonging to multiple classes.

Additional issues. An inherent weakness of C4.5 is that the information gain attribute selection criterion has a tendency to favor many-valued attributes [25]. Quinlan offers a solution to this problem in [26]. Attribute-oriented induction is yet another solution since it results in the generalization of many values into a small set of distinct values.

By creating a branch for each decision attribute value, C4.5 encounters the overbranching problem caused by unnecessary partitioning of the data. Various solutions have been proposed for this problem [6, 20, 27]. MedGenAdjust also addresses this problem since it allows node merging, thereby discouraging overpartitioning of the data.

6. Conclusions and future work

We have proposed a simple but promising method for data classification which addresses efficiency and scalability issues regarding data mining in large databases. We have proposed two algorithms, MedGen and MedGenAdjust. Each algorithm first generalizes the data to an intermediate abstraction level by attribute-oriented induction. This compresses the data while allowing the data to be used at more meaningful levels of abstraction. Relevance analysis is then applied to remove irrelevant data attributes, thereby further compacting the generalized data. The resulting data are much smaller than the original training set. MedGen then applies a modified version of the C4.5 algorithm [26] to extract classification rules from the generalized relation by inducing a decision tree. MedGenAdjust is a refinement of MedGen since it allows for level-adjustment across multiple abstraction levels during decision tree construction. With MedGenAdjust, a balance is struck between user-desired levels and automatic refinement of the abstraction level to promote quality classification.

Generalization of the training data, and the use of a multi-dimensional data cube make the proposed algorithms scalable and efficient since the algorithms can then operate on a smaller, compressed relation, and can take advantage of fast

data accessing due to the indexing structure of the cube.

Our future work involves further refinement of the level-adjustment procedure of MedGenAdjust. For example, we plan to consider the merging of cousin, niece/nephew nodes in addition to the merging of siblings in *node-merge*.

References

- [1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. 18th Intl. Conf. Very Large Data Bases (VLDB)*, pages 560–573, Vancouver, Canada, 1992.
- [2] H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proc. 9th Nat. Conf. on Artificial Intelligence*, volume 2, pages 547–552, Menlo Park, CA, July 1991. AAAI Press.
- [3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification of Regression Trees*. Wadsworth, 1984.
- [4] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.
- [5] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proc. 2nd Intl. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 82–88, Portland, Oregon, 1996.
- [6] U. M. Fayyad. Branching on attribute values in decision tree generation. In *Proc. 1994 AAAI Conf.*, pages 601–606, AAAI Press, 1994.
- [7] U. M. Fayyad, S. G. Djorgovski, and N. Weir. Automating the analysis and cataloging of sky surveys. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 471–493. AAAI/MIT Press, 1996.
- [8] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [9] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.
- [10] D. H. Freeman, Jr. *Applied Categorical Data Analysis*. Marcel Dekker, Inc., New York, NY, 1987.
- [11] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.
- [12] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, 1994.
- [13] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.
- [14] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Intl. Conf. on Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.
- [15] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.
- [16] L. B. Holder. Intermediate decision trees. In *Proc. 14th Intl. Joint Conf. on Artificial Intelligence*, pages 1056–1062, Montreal, Canada, Aug 1995.
- [17] G. H. John. Irrelevant features and the subset selection problem. In W. W. Cohen and H. Hirsh, editors, *Proc. 11th Intl. Conf. on Machine Learning*, pages 121–129, San Francisco, CA, July 1994. Morgan Kaufmann.
- [18] M. Kamber, R. Shinghal, D. L. Collins, G. S. Francis, and A. C. Evans. Model-based 3D segmentation of multiple sclerosis lesions in magnetic resonance brain images. *IEEE Transactions on Medical Imaging*, 14(3):442–453, 1995.
- [19] H. J. Loether and D. G. McTavish. *Descriptive and Inferential Statistics: An Introduction*. Allyn and Bacon, 1993.
- [20] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. 1996 Intl. Conf. on Extending Database Technology (EDBT'96)*, Avignon, France, March 1996.
- [21] J. Mingers. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4(3):227–243, 1989.
- [22] R. Mooney, J. Shavlik, G. Towell, and A. Grove. An experimental comparison of symbolic and connectionist learning algorithms. In *Proc. 11th Intl. Joint Conf. on Artificial Intelligence*, pages 775–787, Detroit, MI, Aug 1989. Morgan Kaufmann.
- [23] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.
- [24] J. R. Quinlan. Learning efficient classification procedures and their application to chess end-games. In M. et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 463–482. Morgan Kaufmann, 1983.
- [25] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [26] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [27] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: a scalable parallel classifier for data mining. In *Proc. 22nd Intl. Conf. Very Large Data Bases (VLDB)*, pages 544–555, Mumbai (Bombay), India, 1996.
- [28] R. Uthurusamy, U. Fayyad, and S. Spangler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–157, Menlo Park, CA, 1991. AAAI/MIT Press.
- [29] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proc. 11th Intl. Conf. on Artificial Intelligence*, pages 781–787, Detroit, MI, Aug 1989. Morgan Kaufmann.
- [30] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufman, 1991.