SSDT: A Scalable Subspace-Splitting Classifier for Biased Data

Haixun Wang
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
haixun@us.ibm.com

Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
psyu@us.ibm.com

Abstract

Decision trees are one of the most extensively used data mining models. Recently, a number of efficient, scalable algorithms for constructing decision trees on large disk-resident dataset have been introduced. In this paper, we study the problem of learning scalable decision trees from datasets with biased class distribution. Our objective is to build decision trees that are more concise and more interpretable while maintaining the scalability of the model. To achieve this, our approach searches for subspace clusters of data cases of the biased class to enable multivariate splittings based on weighted distances to such clusters. In order to build concise and interpretable models, other approaches including multivariate decision trees and association rules, often introduce scalability and performance issues. The SSDT algorithm we present achieves the objective without loss in efficiency, scalability, and accuracy.

1 Introduction

Decision trees are one of the most extensively used data mining models. Decision tree induction is a greedy algorithm that partitions the data in a top-down, divide-and-conquer manner. Decision trees are especially attractive in mining large datasets because i) the decision tree model is easier to interpret [5] and, ii) the induction process is more efficient compared to other methods [12, 8]. Recently, a number of efficient, scalable algorithms for constructing univariate decision trees from large disk-resident data have been introduced [12, 8, 7, 14].

Our work focuses on the scenario where the training data has a highly imbalanced class distribution, i.e., we assume there are only 2 class labels, positive and negative, and the positive (target) class accounts for a small fraction (say between 0.1% and 5%) of the entire dataset. This situation arises frequently in the data mining environment, such as in fraud detection, network intrusion detection, and etc. In this paper, we first discuss several limitations of the decision tree induction process under this scenario. Then, we introduce a new technique, SSDT, which aims at overcoming these problems while preserving the efficiency of the decision tree algorithms.

Representational limitations of univariate decision trees The decision tree induction process has several deficiencies. Consider the training set in Figure 1(a), where `Play?` is the class label. The C4.5 decision tree is shown in Figure 1(b), which has only one node. Obviously, it misses the pattern of `(hot, high)` that has a strong support of `No`. The difficulty is largely due to univariate tests at each node and it is also compounded by the use of categorical attributes.

Applying multivariate decision tree algorithms, such as OC1 [11] and LMDT [6], to such datasets may reveal patterns univariate decision tree can not discover. However, it is computationally expensive to gauge the value of a linear combination of many variables per node, and is practically infeasible to use on large disk-resident datasets.

Biased data distribution Training sets of certain data mining tasks have a very biased data distribution, inasmuch as the target class accounts for an extremely tiny fraction (say 0.5%) of the data. Most learning algorithms, such as decision trees, will generate a trivial model that always predicts the majority class and reports an accuracy rate of 99.5% [16]. However, data cases of the biased class often carries more significant meanings and are often the primary targets of mining.

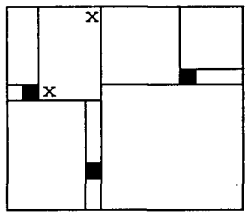
Decision trees have been shown unsuitable for such tasks [10]. In Figure 1(c), we show the distribution of a synthetic dataset in a two-dimensional space, where the dark regions represent the data cases of the biased class. Data cases of the majority class are either clustered at different spots or simply distributed randomly and are not shown explicitly. A representative univariate decision tree learned from such a dataset without pruning is shown in Figure 1(d). The decision tree induction process keeps on partitioning the space either horizontally or vertically at each node, and the resulted decision tree often has a very large size. Furthermore,

0-7695-1119-8/01 $17.00 © 2001 IEEE

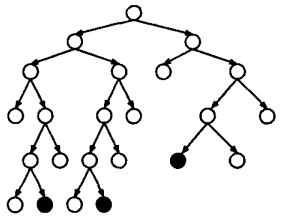
Temp	Humid	Play?
cool	high	yes
mild	high	yes
mild	normal	yes
hot	dry	yes
hot	normal	yes
hot	normal	yes
hot	high	no
hot	high	no
hot	high	no
very hot	high	yes

(a) Training Set

(b) C4.5 decision tree for (a)



(c) 2-dimensional biased dataset



(d) decision tree for (c)

Figure 1. Limitations of Decision Tree Classifiers

it is easy to see that the size of the tree will grow with the dimensionality of the data, as in a higher dimensional space more tests are required to locate the subspace region where the data cases cluster.

A decision tree that assigns an unknown instance a definite class label ('positive' or 'negative') or a probability based merely on the data distribution in the leaf nodes usually has a low predictive accuracy for instances of the biased target class. Furthermore, the model can hardly discriminate among data cases of the majority class on the basis of their closeness to the target class. For instance, the two 'X's in Figure 1(c) are in the same leaf node. Hence, they are assigned a same probability by the model, although one of them is much closer to the biased class. This causes problems in applications such as target marketing: the marketing department wants to send out promotions to 20% of the people in their database while only 1% of the people in the database are recorded buyers. The rest 19% has to be selected based on their closeness to the buyer class.

2 Our Approach: An Overview

It is clear from the examples in Figure 1 that univariate splitting conditions often results in undetected patterns or decision trees of formidable sizes. To build concise models, it is essential that more than one variables are taken into consideration in splitting the data. However, scalability requirements forbids us considering all possible combinations of these variables as OC1 [11] and LMDT [6] do.

We improve the decision tree induction process by providing an additional splitting criterion which results in more concise and more interpretable decision trees. The motivation is that, in datasets with biased class distribution, while the negative instances are distributed 'everywhere', instances of the target class tend to form clusters in some subspaces. These clusters are the foundation of accurate predictions of unknown instances. The proposed approach, SSDT, uses an efficient multivariate search algorithm to locate subspace clusters so as to enable (multivariate) splitting based on weighted distances to the centroid of the clusters.

While it is computationally prohibitive to search for all the clusters, it is more feasible to search for clusters formed by points in the biased target class, since they only account for a small fraction of the data. Our algorithm detects candidate clusters from lower dimensions to higher dimensions, and prunes away all the candidates as soon as we find that partitions based on these clusters can not offer a better purity than the univariate splits.

Related Works Much work has focused on how to tackle the deficiencies of the decision tree discussed in the previous section. Among them, multivariate decision trees overcome a representational limitation of univariate decision trees [13, 4]. However, performing a multivariate partition often leads to a much larger consumption of computation time and memory, which may be prohibitive for large datasets.

The target selection problem also attracts lots of attention [9, 10]. Closeness estimations of an unknown instance to a certain class can be solved by clustering and nearest neighbor algorithms. A naive approach searches for clusters of the biased data and scores an unknown sample by its distance to the closest cluster. Association rule mining [3] is also used to solve the target selection problem. A potential problem with association rules is the combinatorial explosion of "frequent itemsets" [9, 10], which can be prohibitive for large datasets with biased data distribution even after sophisticated pruning.

Contributions of this Paper We use novel splitting criteria in building decision trees. We discover data clustering in correlated dimensions and partition the data by distance functions defined in the corresponding subspaces. With our multivariate splitting condition, decision trees can be built

smaller, and more interpretable. However, unlike other multivariate decision tree algorithms that are usually prohibitive for large datasets or high dimensions, our approach is efficient and scalable.

3 Definitions

Let S denote the training set belonging to a node of a decision tree. Each point in S has n attributes in addition to the special attribute: class label. Let $C = \{x_1, ..., x_k\}$ be a cluster of points where each x_i has the same class label. The *centroid* of C is the algebraic mean of points in C.

To measure the closeness of a point to a cluster, we use weighted Euclidean distance function, $Dist(\vec{d}, \vec{p}, \vec{w}) = \sqrt{\sum_{i=1}^{n} \vec{w}_i(\vec{d}_i - \vec{p}_i)^2}$, where $\vec{d}$ is a point, $\vec{p}$ the centroid of a cluster, and $\vec{w}$ the weight vector. The Euclidean distance is indeed defined in a subspace that is specified by a set of dimensions whose weights are non-zero.

The Euclidean distance only works for numerical attributes. For ordinal attributes, we can map their values to the range of [0,1]. For categorical attributes, the heterogeneous Euclidean metric [15] defines the similarity of two values by their relative frequency of occurrences in the same class. However, for attributes with many values and a biased dataset, certain values may never occur in the training set. In our algorithm, we use a distance matrix M supplied by the user, such that the value $M(i,j) \in [0,1]$ denotes the distance between categorical value i and j.

We use the *gini* index,

$$gini(S) = 1 - \sum_{j=1}^{c} p_j^2 \quad (1)$$

where p_j is the relative frequency of class j in S, to measure the "goodness" of all the potential splits. If S is partitioned into two subsets S_1 and S_2, the index of the partitioned data $gini(S)$ can be obtained by:

$$gini(S) = \frac{n_1}{n_1 + n_2} gini(S_1) + \frac{n_2}{n_1 + n_2} gini(S_2) \quad (2)$$

where n_1 and n_2 are the number of points of S_1 and S_2, respectively.

4 Scalable Decision Tree Classifiers

A decision tree classifier recursively partitions the training set until each partition consists entirely, or almost entirely, of records from one class.

The SPRINT algorithm has been proposed to build decision trees for large datasets [12]. The splitting criterion used by SPRINT is based on the value of a single attribute (univariate). For a continuous attribute, it has the form of $A \leq C$ where A is an attribute and C is a value in the domain of attribute A.

SPRINT avoids costly sorting at each node by presorting continuous attributes only once, at the beginning of the algorithm. Values of each continuous attribute are maintained by a sorted list. Each entry in the list contains i) a value of the attribute, ii) its record id (*rid*), and iii) the class label of the record. The node is split on the attribute which yields the least value of the *gini* index (G_{best}). Based on the sorted list of the splitting attribute, a hash table is constructed to map each record (*rid*) to one of the subnodes which the record belongs to after the split. Entries in other attribute lists are moved to the attribute list of the subnodes after consulting the hash table as to which subnode this entry belongs to. The sorted order is maintained as the entries are moved in pre-sorted order.

5 The SSDT Algorithm

The core of SSDT lies in detecting subspace clusters of positive points. However, finding all such clusters is both time consuming and unnecessary. We are only interested in clusters that can offer a better split than univariate partitions. In Section 5.1, we prove an important property which enables us to narrow down our search to those clusters that have the potential. The actual clustering algorithm is introduced in Section 5.2, where we use an Apriori-like algorithm to find subspace clusters from lower dimensions spaces to higher dimensional spaces. In Section 5.3, we compute the exact *gini* index for cluster-based partitioning by scanning a small proportion of the data, thus keeping the overhead of the multivariate partitioning to a minimum.

SSDT is based on the framework of SPRINT, where presorted attribute lists are maintained at each node. We consider only two class labels, `positive` and `negative`, and the target class (`positive`) is biased, usually accounting for only a small fraction of the data. We normalize the values on each dimension to the range of [0,1].

The SSDT approach is outlined in Algorithm 1. To partition a dataset, we first compute the *gini* index on each of its attributes. While we scan through the pre-sorted attribute list, we also derive 1-dimensional clusters of the biased data for each dimension (described in detail later). Next, we locate subspace clusters of the positive data cases. The minimal *gini* index produced by the univariate splits, G_{best}, is passed in as a parameter to $ClusterDetect()$ so that clusters can not possibly deliver a *gini* index smaller than G_{best} are pruned as early as possible. We then compute the *gini* index of splits based on the distance to each subspace cluster. The process, $DistanceEntropy()$, described in Algorithm 2, does not require globally reordering the data according to the distance. If the minimal *gini* index is achieved by some subspace cluster, we partition

Algorithm 1 SSDT(Dataset: S)

1: $S_p \leftarrow$ points of the biased class in S;
2: **for each** attribute k **do**
3: scan the sorted attribute list of k and compute:
4: — I_k : the *gini* index on attribute k;
5: — L_k : clusters of points in S_p on attribute k;
6: **end for**
7: $G_{best} \leftarrow \min_k I_k$;
8: $\mathcal{C} \leftarrow ClusterDetect(G_{best}, S_p, L)$;
9: **for each** cluster $c \in \mathcal{C}$ **do**
10: $I'_c \leftarrow DistanceEntropy(c, S)$;
11: **end for**
12: **if** $\min_{c \in \mathcal{C}} I'_c < G_{best}$ **then**
13: split S into two subsets S_1, S_2 based on the distance;
14: **else**
15: split S into subsets on the attribute with G_{best};
16: **end if**
17: call SSDT(S_i) on each subset S_i if S_i does not satisfy the termination condition;

the dataset based on the distance to such a cluster. More specifically, given a point $\vec{d}$, instead of using a univariate test $\vec{d_i} \leq v$, we use a test in the form of $Dist(\vec{d}, \vec{p}, \vec{w}) \leq v$, where $\vec{p}$ is the centroid of the cluster and $\vec{w}$ is a weight vector of all the dimensions. As in the SPRINT algorithm, the partition process also keeps the sorted order of the attribute lists, so that no reordering is required. The partition stops when a node is composed entirely of negative points (100%) or almost entirely[1] of positive points.

5.1 Minimal Support of Subspace Clusters

To find subspace clusters of points (of the biased class), we need to find: i) the centroid $\vec{p}$ of the cluster, and ii) the weight $\vec{w}$ which defines the subspace ($\vec{w}_i = 0$ means dimension i is irrelevant) of the cluster. Several subspace clustering algorithms have been introduced in the literature. The CLIQUE algorithm [2] reports connected dense units in subspaces but the centroids of clusters are not detected. Another method, called PROCLUS [1], uses a hill climbing method to successively improve a set of centroids, and derives a set of dimensions for each cluster. This algorithm however, requires that the number of clusters k to be found is pre-known. Both methods are time consuming since they aim at discovering all the subspace clusters.

Given a found cluster, Algorithm 1 partitions a dataset S into 2 datasets, S_1 and S_2, such that S_1 contains points close to the centroid of the cluster. Instead of checking all the subspace clusters, we are only interested in those that can result in partitions with a *gini* index lower than G_{best}, the minimal gini index we get by partitioning the data on single attributes.

Proposition 1 tells us how to narrow our search on qualified clusters. We define the *support* of a cluster as P'/P, where P' is the number of (positive) points in the cluster, and P is the total number of positive points in S. We prove the following proposition:

Proposition 1. *If the gini index of a cluster-based partition of S is lower than G_{best}, then the cluster must have a support greater than $\frac{2q-2q^2-G_{best}}{2q-2q^2-qG_{best}}$, where q is the percentage of the positive points in S.*

Proof. Let N be the total number of points in S. Let P be the total number of positive points in S. Thus, $q = P/N$. Assume S is partitioned into S_1 and S_2, and S_1 contains the points in the cluster. According to Formula 2, we have:

$$gini(S) = \frac{P' + N'}{N} gini(S_1) + \frac{N - P' - N'}{N} gini(S_2)$$

where P' and N' are the number of positive points and negative points in S_1 respectively. Given $N \gg P'$, it can be shown that the lowest $gini(S)$ is achieved if S_1 contains only positive points, that is, $N' = 0$ and $gini(S_1) = 0$. That the partition produces a *gini* index lower than G_{best} means:

$$\frac{N - P'}{N} gini(S_2) < G_{best}$$

Expanding $gini(S_2)$ using Equation 1, we get:

$$\frac{-2NP' - 2P^2 + 2PP' + 2NP}{N(N - P')} < G_{best}$$

Substituting P/N with q, we get:

$$minsup = P'/P > \frac{2q - 2q^2 - G_{best}}{2q - 2q^2 - qG_{best}} \qquad (3)$$

The *minsup* given by Formula 3 is a lower bound, because the cluster we find usually does not contain only positive points (i.e., $N' > 0$). □

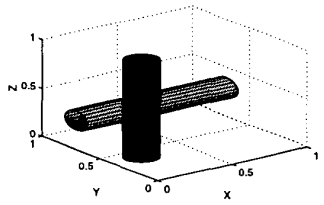
5.2 Cluster Detection

To find subspace clusters of points in the biased class, we use an iterative approach that is very similar to the apriori algorithm [3] for finding frequent itemsets. A cluster whose support is lower than *minsup* in k-dimensional space can not have support larger than *minsup* in $(k+1)$-dimensional space. We first find clusters in 1-dimensional spaces, then we combine them to form candidate clusters

[1] We assign a higher weight to each positive point to balance the biased distribution. In our algorithm, we stop if more than 90% of the points in the node is positive.

in 2-dimensional spaces. We count the number of points in each cluster and eliminate those candidate clusters whose support does not satisfy the constraint in Proposition 1. Then we combine clusters in the 2-dimensional spaces to form candidates in the 3-dimensional spaces, and so on, until no more qualified clusters can be found.

Figure 5.2(a) shows an example where points of the biased class form two clusters in a 3-dimensional space. We use a simple approach to detect 1-dimensional clusters. In Figure 3, the range of each attribute is divided into 10 bins and we keep the counts of points that fall in each bin. This is done when we scan through attribute lists to evaluate splits on single attributes, so there is minimal extra cost introduced. The horizontal line in Figure 3 indicates the average density and we regard each continuous region above the average density line as *one* cluster. Thus, we detect one cluster around .1 with radius .1 on attribute X, two clusters around .3 and .7 respectively both with radius .1 on attribute Y, and one cluster around .2 with radius .1 on attribute Z.

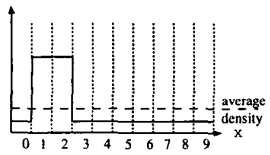


(a) Points of biased class form two clusters

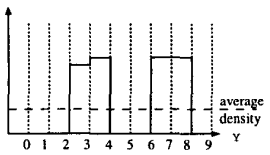
X	Y	Z
.1/.1	.3/.1	-
.1/.1	.7/.1	-
.1/.1	-	.2/.1
-	.3/.1	.2/.1
-	.7/.1	.2/.1

(b) Potential centroids of clusters

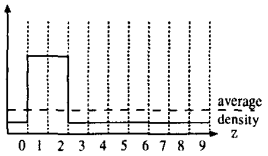
Figure 2. Cluster detection



(a) Density of Points on X



(b) Density of Points on Y



(c) Density of Points on Z

Figure 3. Histograms of positive points on each attribute

Assuming all the 1-dimensional clusters shown in Figure 3 has support larger than *minsup*, we then form a list of potential centroids in the 2-dimensional subspace as shown in Figure 5.2(b). Each centroid is represented by values on two dimensions only, $(..., c_i/r_i, ..., c_j/r_j, ...)$, where c_i and c_j are centers of clusters on dimension i and j respectively, and r_i and r_j are their radius. Values on the other dimensions are unknown. Next, we make one scan through all the points in the biased class: for all points $\vec{d}$, $\vec{d} \in C = \{\vec{d} | r_i \geq |\vec{d_i} - c_i|, r_j \geq |\vec{d_j} - c_j|\}$, we compute the mean $c_k = \sum_{\vec{d} \in C} \vec{d_k}/|C|$, and the radius r_k on each of the dimension k.

Algorithm 2 ClusterDetect(GiniIndex: G_{best}, Biased Data: S_p, Center/Radius Lists: L)

```
1: minsup ← derived by G_best (Formula 3);
2: l ← 2; C_l ← ∅;
   {Step 1. find clusters in 2-dimensional space}
3: for each center/radius c_i/r_i in L_i do
4:    for each center/radius c_j/r_j in L_j, j > i do
5:       add (..., c_i/r_i, ..., c_j/r_j, ...) to C_l;
6:    end for
7: end for
   {Step 2. find clusters from lower to higher dimensions}
8: while C_l ≠ ∅ do
9:    scan the points in S_p and increase the count of cluster
      c ∈ C_l for each point that belongs to c;
10:   eliminate cluster c from C_l if the number of points in
      c is less than minsup;
11:   l ← l + 1;
12:   C_l ← combining clusters in dimension l − 1;
13: end while
   {Step 3. return the clusters}
14: return top-k leaf clusters;
```

After clusters whose support is less than the value given by Formula 3 are pruned, we explore clusters in higher dimensional spaces and repeat this process until no more clusters can be found. Finally, $ClusterDetect()$ returns the found clusters. We are only interested in *leaf* clusters, which are clusters that do not contain other clusters. Among all the leaf clusters, we return the top K clusters in the higher dimensions, where K is a user-specified parameter.

In order to compute the weighted Euclidean distance between a point and a cluster, we need to find out the weight on each dimension. For a non-clustered dimension i, we set $\vec{w_i} = 0$; otherwise, we set $\vec{w_i} = 1/r_i^2$, where r_i is the radius of the points' distribution on dimension i. Thus, the distance is normalized to reflect the span of the points on each dimension.

5.3 Split by Distance

For each cluster returned by $ClusterDetect()$, we derived a distance function $Dist(\vec{d}, \vec{p}, \vec{w})$. The next step is to

find the value v so that the split by the test $Dist(\vec{d}, \vec{p}, \vec{w}) \leq v$ offers the minimum *gini* index.

A straight-forward approach is to reorder all the points by their distances to the center $\vec{p}$, and compute the *gini* index by scanning the ordered points. This is costly for large datasets. Another approach is to discretize the distance into intervals and for each interval we keep the counts of positive/negative cases whose distance to $\vec{p}$ are in that interval. One shortcoming of this approach is the loss of accuracy due to discretization.

Our approach, outlined in Algorithm 3, avoids reordering all the data and any loss of accuracy. This is achieved by making the following two observations: i) if point $\vec{d}$ is close to centroid $\vec{p}$, then $\vec{d}_i$, the coordinate on the i-th dimension, must also be close to $\vec{p}_i$; and ii) the best splitting position should be close to the boundary of the cluster.

Let N be the number of dimensions with non-zero weights (clustered dimensions). Let D be the set of points that are within an initial radius $r = \delta$ to $\vec{p}$. For any point $\vec{d} \in D$, the inequality $\vec{w}_i(\vec{d}_i - \vec{p}_i)^2 \leq r^2$ must hold for each clustered dimension i. With the ordered attribute lists, it is easy to find D', points that satisfy the inequality on all the N clustered dimensions. Obviously $D' \supseteq D$, for D' can contain points whose distance to $\vec{p}$ is up to $r\sqrt{N}$. After sorting D' by distance, we compute the *gini* index up to radius r, and we keep the points in $D' - D$ and discard D.

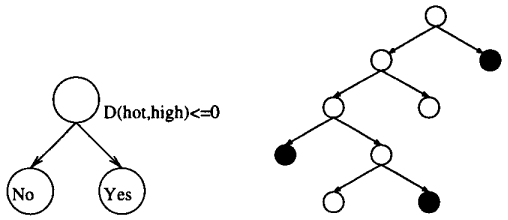
We then increase the radius r by δ and repeat the process. However, we do not have to consider all the points. We are computing the *gini* index based on the distance to the cluster centroid we found. Thus, we expect a good *gini* index near the boundaries of the cluster. According to the weighting scheme discussed in the previous subsection, $\vec{w}_i$ is set to $1/r_i^2$ for each clustered dimension i, where r_i is the span of the points on that dimension. Thus, we have $\vec{w}_i(\vec{d}_i - \vec{p}_i)^2 \leq 1$ for any point $\vec{p}$ that is inside the cluster. In addition to these points, we consider all points that satisfy $\vec{w}_i(\vec{d}_i - \vec{p}_i)^2 \leq 2$ on each dimension i. Thus, the maximum radius of r is $\sqrt{\sum_i \vec{w}_i(\vec{d}_i - \vec{p}_i)^2} \leq \sqrt{2N}$.

5.4 SSDT Examples

Let us review the two problems in Section 1. Unlike the C4.5 decision tree, which fails to detect pattern `(hot,high)` and builds a trivial decision tree in Figure 1(b) , the SSDT algorithm accurately captures the pattern and constructs a compact decision tree in Figure 4(a). The second problem is introduced by datasets with biased class distribution. The decision tree model shown in Figure 1(d) used 11 tests to classify a 2-dimensional dataset shown in Figure 1(c). SSDT, shown in Figure 4(b), uses only 4 tests. Apparently, such differences tend to be more significant if the dataset has more than 2 dimensions.

Algorithm 3 distance_entropy(Dataset: S, Centroid: $\vec{p}$, Weight: $\vec{w}$)

```
1: r ← δ;
2: N ← # of dimensions with non-zero weights;
3: repeat
4:   for each relevant dimension i do
5:     find instances d⃗ that satisfies (r − δ)² ≤ w⃗_i(d⃗_i − p⃗_i)² < r² using the ordered attribute lists;
6:     d⃗.count ← d⃗.count + 1;
       {check if d⃗ satisfies all the inequalities}
7:     add d⃗ to the ordered set D' if d⃗.count = N;
8:   end for
9:   compute gini-index for splits by distance up to r;
10:  remove in tree D' the branch that represents data cases within distance r to p⃗;
11:  r ← r + δ;
12: until r > √(2N);
13: return I' and v;
```



(a) SSDT of Figure 1(a) (b) SSDT of Figure 1(c)

Figure 4. SSDT for datasets in Section 1

6 Evaluations

We evaluate the SSDT algorithm in various aspects. We study the size of the decision tree generated by the algorithm, the influence of the biased class distribution, the accuracy of predictions, as well as the efficiency and scalability issues. The tests were performed on a 700-MHz Pentium III machine with 256M of memory, running Linux.

Synthetic Data Generation We generate synthetic data in d-dimensional spaces with two class labels, positive and negative. Points have coordinates in the range of [0,1] and positive points account for $p = 1\% - 10\%$ of the total data.

To generate clustered points in subspaces, we use a method similar to [1]. The difference is that the number of positive points is controlled by the biased class ratio. Our method takes 4 parameters: n, the number of clusters; k, a Poisson parameter that determines the number of relevant dimensions in each cluster; p, the percentage (biased ratio) of positive points, and N, the total number of points.

First we determine the subspace for each cluster. For a given cluster i, the number of relevant dimensions, S_i, is

picked from a Poisson distribution with mean k. However, an additional restriction, $2 \leq S_i \leq d$, must be observed.

We generate centroid $\vec{p}_i$ for each cluster i. We simply generate a uniformly distributed point in the d-dimensional space. We then decide the spread (radius) of the cluster on each dimension. We set $\vec{r}_{ij} = 0.5$ for irrelevant dimension j. For a clustered dimension, we fix a *spread parameter* s and choose the radius $\vec{r}_{ij} \in [0, s]$ uniformly at random. For our data generation, we use 3 values for $s = .1, .2, .5$.

We generate positive points in each cluster i in two different ways: i) points are distributed uniformly in the region; ii) for each dimension j, coordinates of points on the dimension follow a normal distribution with mean $\vec{c}_{ij}$ and variance $\vec{r}_{ij}^2$. We determine the size of each positive cluster by $N_i = pN\frac{v_i}{\sum_{i=1}^{n} v_i}$, where v_i is the volume of cluster i defined as $v_i = \prod_{j=1}^{d}(2 * r_{ij})$.

Finally, we generate $(1 - p)N$ negative points. The negative points either i) uniformly distribute at random in the entire space, or ii) form clusters in subspaces and are generated by the method described above with parameters $k = 0.8d$ and $s = 0.5$. If a negative point is generated inside one of the positive clusters, it is discarded with a probability δ. For our data generation, we choose $\delta = 0.5$.

Experiments: Tree Size and Scalability We generate 6 clusters (Table 1) of positive points out of a training set of total 100K records and 10 attributes. The total number of positive cases account for 2% of the training set. The average radius of the cluster on each clustered dimension is 0.05, and the negative points are uniformly distributed at random. The split at the root of the decision tree, for example, uses the distance function defined for Cluster 6. Total 5 clusters are used at different nodes for splitting, resulting in a tree of 37 leaf nodes before pruning, while the SPRINT algorithm uses 71 leaf nodes before pruning.

Cluster	Centroids	Points
1	(-, 0.35, -, -,-,0.62,-,0.77,-,0.26,0.27)	87
2	(-,0.74,-,0.11, -,0.85,-,-,-,-)	199
3	(0.92, 0.22, -, -, 0.81, -, -, -, -, -)	204
4	(-, -, 0.37, 0.12, -,-,0.63,-,-,-,-)	212
5	(-,-,-,-,-,0.32, 0.20,0.14,0.87,0.43)	83
6	(-,-,0.37,-,-,-,-,0.66,-,-)	1211

Cluster	Centroid Detected by SSDT	Points
6	(-,-,0.37,-,-,-,-,0.66,-,-)	1228
4	(-, -, 0.37, 0.11, -,-,0.62,-,-,-,-)	186
2	(-,0.74,-,0.13, -,0.85,-,-,-,-)	184
3	(0.92, 0.22, -, -, 0.81, -, -, -, -, -)	1124
1	(-, 0.34, -, -,-,0.62,-,0.74,-,0.26,0.27)	83

Table 1. 5 clusters are detected by SSDT. Unclustered dimensions are denoted by '-'.

We compare the size of the decision tree (in terms of number of leaf nodes) generated by SPRINT and SSDT. The datasets we use have 10 attributes, and the 5 clusters of biased points account for 1%, 2% and 5% of the total data. Figure 6(a) indicates that trees built by SSDT are significantly smaller, and the sizes of the trees generally do not increase as the training sets become larger.

Next, we vary the number of clusters in the training sets and show the results in Figure 6(b). The datasets are generated with the same class ratio: 2%. The size of the tree increases significantly as there are more positive clusters in the dataset. The trees generated by SSDT are much smaller.

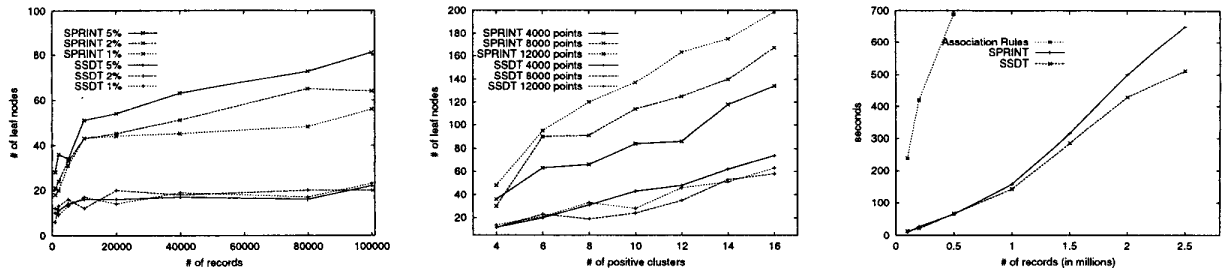
Figure 6(a) shows the scalability of SSDT as the size of the dataset increases from 0.1 to 2.5 million. The dataset has 10 attributes, 8 clusters with an average dimensionality of 4, and a biased class ration of 1%. The execution time increases linearly with the size of the dataset, since SSDT is able to detect the clusters and the resulted decision tree has similar heights, which means the number of passes through the database does not change. Figure 6(b) shows the scalability of SSDT when the average dimensionality of the positive clusters is increased from 2 to 12. The dataset used in the test has 1 million records, 8 clusters, 1% positive ratio, and a total of 20 attributes. It indicates that cluster dimensionality has little impact on the performance.

We study the impact of the number of positive clusters on the scalability. In Figure 6(c), we increase clusters from 4 to 20. The dataset has 1 million records, among which 1% are positive. There are 10 attributes and the clusters have an average of 5 dimensions. Since the number of positive data cases is kept unchanged during the test, each cluster contains fewer records as more clusters are used. The curve is steeper than in the previous cases because more scans of the dataset have to be performed. In Figure 6(d), using the dataset of the same size, dimensionality, and 8 positive clusters, we found the performance is stable.

We compare the performance of SSDT with SPRINT in Figure 6(c). The datasets have 10 attributes, 8 positive clusters, and a class ratio of 1%. In this case, SSDT has an advantage over SPRINT because SSDT trees are much smaller. As we increase the class ratio and the number of clusters, SPRINT becomes faster than SSDT. Indeed, SPRINT is 20% faster than SSDT when there are 20 clusters with a 15% class ratio, which means SSDT works best with biased class distributions. The association rule algorithm for mining datasets with biased class distribution does not scale well. Overall, SSDT is an efficient and scalable algorithm, despite the multivariate search it performs.

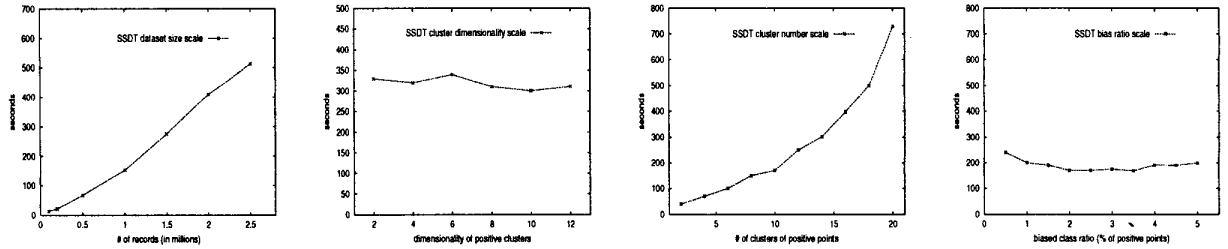
7 Conclusion

We presented a novel decision tree algorithm. The key idea is to take advantage of the subspace clusters formed by



(a) # of leaf nodes v. training set size (b) # of leaf nodes v. # of clusters (c) Execution Time

Figure 5. Experiments and Comparisons



(a) # of records (b) dimensionality (c) # of clusters (d) class distribution

Figure 6. Scalability

the data in the biased class. Once these subspace clusters are efficiently detected, a compact and accurate decision tree can be constructed by splitting a node based on the distance to such clusters. Our multivariate decision tree algorithm has proven to be scalable and efficient. Indeed, it has better performance over SPRINT for very skewed distributions.

References

[1] C. C. Aggarwal, C. Procopiuc, J. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD*, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Authomatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[4] J. Bioch, O. van der Meer, and R. Potharst. Bivariate decision trees. In *Principles of Data Mining and Knowledge Discovery*, 1997.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[6] C. E. Brodley and P. E. Utgoff. Multivariate versus univariate decision trees. In *Technical Report COINS-CR-92-8, Dept. of Computer Science, University of Massachusetts*, 1992.

[7] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat–optimistic decision tree construction. In *SIGMOD*, 1999.

[8] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest: A framework for fast decision tree constructionof large datasets. In *VLDB*, 1998.

[9] G. Guiffrida, W. W. Chu, and D. M. Hanssens. Mining classification rules from datasets with large number of many-valued attributes. In *EDBT*, pages 335–349, 2000.

[10] Y. Ma, B. Liu, C. K. Wong, P. S. Yu, and S. M. Lee. Targeting the right students using data mining. In *SIGKDD*, Zurich, Switzerland, August 2000.

[11] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. In *Journal of Artificial Intelligence Research*, pages 1–32, 1994.

[12] C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *VLDB*, 1996.

[13] P. E. Utgoff and C. E. Brodley. An incremental method for finding multivariate splits for decision trees. In *ICML*, pages 58–65, 1990.

[14] H. Wang and C. Zaniolo. CMP: A fast decision tree classifier using multivariate predictions. In *ICDE*, pages 449–460, 2000.

[15] D. Wilson and T. Martinez. Improved heterogeneous distance functions. In *Journal of Artificial Intelligence Research*, pages 1–34, 1997.

[16] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Technical Report CS2001-0664, Dept. of Computer Sci., UCSD*, 2001.