

Enhanced SMART-TV: A Classifier with Vertical Data Structure and Dimensional Projections Pruning

Taufik Fuadi Abidin

**Data Mining and IR Research Group
Department of Mathematics, Syiah Kuala University
Banda Aceh, NAD 23111 Indonesia
taufik.abidin@unsyah.ac.id**

William Perrizo

**Computer Science Department
North Dakota State University
Fargo, ND 58105 USA
william.perrizo@ndsu.edu**

Abstract—Classification is the process of assigning class membership to the newly encountered and unlabeled objects based on some notion of closeness between the objects in the training set and the new objects. In this work, we introduce the enhancement of SMART-TV (Small Absolute difference of ToTal Variation) classifier by introducing dimensional projections process to prune the neighbors in the candidate set that are far away from the unclassified object in terms of distance but are close in terms of total variation. SMART-TV is scalable nearest-neighbors classifier. It uses vertical data structure and approximates a set of potential candidate of neighbors by means of vertical total variation. The total variation of a set of objects about a given object is computed using an efficient and scalable Vertical Set Squared Distance algorithm. In our previous work of SMART-TV, the dimensional projections was not introduced, and thus, the candidate set that are far away from the unclassified object but close in terms of total variation also voted. This, in some cases, can reduce the accuracy of the algorithm. Our empirical results using both real and synthetic datasets show that the proposed dimensional projections effectively prune the superfluous neighbors that are far away from the unclassified object. In terms of speed and scalability, the enhanced SMART-TV is very fast and comparable to the other vertical nearest neighbor algorithms, and in terms of accuracy, the enhanced SMART-TV with dimensional projections pruning is very comparable to that of KNN classifier.

Index Terms—Data Mining, Classification, Vertical Data Structure, Vertical Set Squared Distance.

I. INTRODUCTION

Classification on massive datasets has become one of the most important research challenges in data mining. Classification involves predicting the class label of newly encountered objects using feature attributes of a set of pre-classified objects. The main goal of classification is to determine data membership of the unclassified objects and to discover the group of the new objects.

K-nearest neighbor (KNN) classifier is one of the methods

used for classification [5]. KNN classifier does not build a model in advance like decision tree induction, Neural Network, and Support Vector Machine; instead, it invests all the efforts when new objects arrive. The class assignment is made locally based on the features of the new objects. KNN classifier searches for the most similar objects in the training set and assigns class labels to the new instances based on the plurality of class of the k-nearest neighbors. The notion of closeness between the training objects and the new object is determined using a distance measure, e.g. Euclidian distance. KNN has shown good performance on various datasets. However, when the training set is very large, i.e. millions of objects, it will not scale. The brute force searches to find the k-nearest neighbors will increase the classification time significantly.

Several new approaches have been developed to accelerate the classification process by improving the KNN search, such as [9]-[16] just to mention a few. Grother [9] accelerated the nearest neighbor search using kd-tree method, while Vries [16] improved the efficiency of KNN search in high dimensional spaces as a physical database design problem.

Most of the state-of-the-art classification algorithms represent the data as the traditional horizontal row based structuring. This structure, combined with sequential scan-based approach, is known to scale poorly to very large datasets. Gray [8] in his talk at SIGMOD 2004 conference emphasized that vertical column based structuring as opposed to horizontal row based structuring is a good alternative to speed up query process and ensure scalability.

In this work, we introduce the enhancement of SMART-TV classifier with dimensional projections to prune the neighbors in the candidate set that are distance away from the unclassified object, even though in terms of absolute difference of total variation, their values can be small. SMART-TV is an efficient and scalable nearest-neighbors classifier that approximates a set of candidates of neighbors by means of vertical total variation [1]. The total variation of a set of objects about a given object is computed using an efficient and scalable Vertical Set Squared Distance algorithm [2].

Manuscript received September 30, 2009.

T. Abidin is the faculty member of Syiah Kuala University and the founder of Data Mining and Information Retrieval Research Group at the Department of Mathematics, College of Science (e-mail: taufik.abidin@unsyah.ac.id).

W. Perrizo is a Professor of Computer Science at North Dakota State University, Fargo, ND, 58105, USA (e-mail: william.perrizo@ndsu.edu).

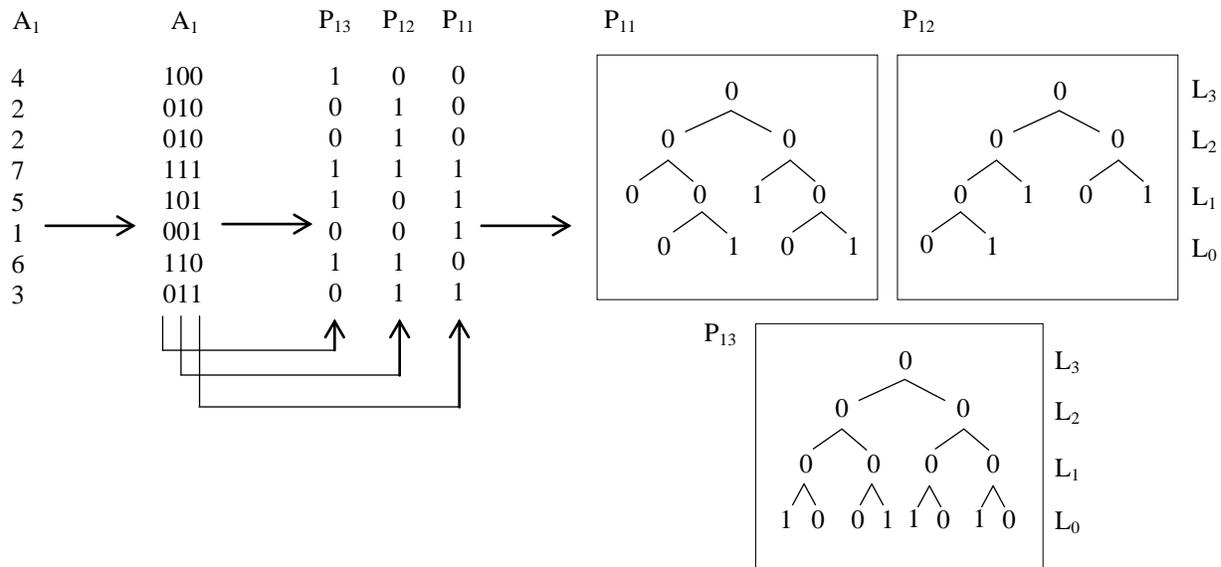


Fig. 1. The construction of P-tree

We use vertical data structure that has been experimentally proven to address the curse of scalability and to facilitate efficient data mining over large datasets. We demonstrate the scalability of the algorithm empirically through several experiments.

II. P-TREE VERTICAL DATA STRUCTURE

Vertical data representation arranges and processes the data differently from horizontal data representation. In vertical data representation, the data is structured column-by-column and processed horizontally through logical operation like AND or OR, while in horizontal data representation, the data is structured row-by-row and processed vertically through scanning or using some indexes.

P-tree vertical data structure is one choice of vertical data representation [13]. P-tree is a lossless, compressed, and data-mining ready data structure. P-tree is lossless because the vertical bit-wise partitioning guarantees that the original data values can be retained completely. P-tree is compressed because when the segments of bit sequences are either pure-1 or pure-0, they can be represented in a single bit. P-tree is data-mining ready because it addresses the curse of cardinality or the curse of scalability, one of the major issues in data mining. P-tree vertical data structure has been intensively exploited in various domains and data mining algorithms, ranging from classification [1]-[2]-[11], clustering [12], association rule mining [14], and outlier analysis [15]. In September 2005, P-tree technology was patented in the United States by North Dakota State University (US patent number: 6,941,303).

The construction of P-tree vertical data structure is began by converting the dataset, normally arranged in a relation of horizontal records, into binary. Each attribute in the relation is vertically partitioned, and for each bit position in the attribute,

vertical bit sequences (containing 0s and 1s) are subsequently created. During partitioning, the relative order of the data is retained to ensure convertibility. In 0-dimensional P-trees, the vertical bit sequences are left uncompressed and are not constructed into predicate trees. The size of 0-dimensional P-trees is equal to the cardinality of the dataset. In 1-dimensional compressed P-trees, the vertical bit sequences are constructed into predicate trees. In this compressed form, AND operations can be accelerated. The 1-dimensional P-trees are constructed by recursively halving the vertical bit sequences and recording the truth of “purely 1-bits” predicate in each half. A predicate 1 indicates that the bits in that half are all 1s, and a predicate 0 indicates otherwise.

The complete set of (uncompressed) vertical bit slices (for numeric attributes) or bitmaps (for categorical attributes) will be called the 0-dimensional P-trees for the given relation of horizontal records. A 1-dimensional P-tree set involves a binary tree compression of those 0-dimensional P-trees as shown in the Fig. 1. A 2-dimensional, 3-dimensional or multi-dimensional P-tree set can be created as well, depending upon the nature of the dataset. In this work, we used the uncompressed P-tree set.

In P-tree data structure, range queries, values, or any other patterns can be obtained using a combination of Boolean algebra operators AND, OR, and NOT. Beside those operators, COUNT operation is also very important in this structure, which counts the number of 1-bits in the basic or complement P-tree. Ding [7] shows the complete description about P-tree algebra and its operations.

III. VERTICAL SET SQUARED DISTANCE

Let $R(A_1, A_2, \dots, A_d)$ be a relation in d -dimensional space. A numerical value v of attribute A_i can be written in b bits binary representation to be:

$$v_{ib-1} \dots v_{i0} = \sum_{j=b-1}^0 2^j \cdot v_{ij} \quad (1)$$

Where v_{ij} can either be 0 or 1. Now, let x be a vector in d -dimensional space. The binary representation of x in b bits can be written as:

$$x = (x_{1(b-1)} \dots x_{10}, x_{2(b-1)} \dots x_{20}, \dots, x_{d(b-1)} \dots x_{d0}) \quad (2)$$

Let X be a set of vectors in R or X be the relation R itself, $x \in X$, and a be the input vector. The total variation of X about a , denoted as $TV(X, a)$, measures the cumulative squared separation of the vectors in X about a , which can be computed quickly and scalably using the Vertical Set Squared Distance (VSSD) formula, derived as follows:

$$TV(X, a) = (X - a) \circ (X - a) = \sum_{x \in X} (x - a) \circ (x - a) = \sum_{x \in X} \sum_{i=1}^d (x_i - a_i)^2$$

$$\begin{aligned} &= \sum_{x \in X} \left(\sum_{i=1}^d x_i^2 - 2 \cdot \sum_{i=1}^d x_i a_i + \sum_{i=1}^d a_i^2 \right) \\ &= \sum_{x \in X} \sum_{i=1}^d x_i^2 - 2 \cdot \sum_{x \in X} \sum_{i=1}^d x_i a_i + \sum_{x \in X} \sum_{i=1}^d a_i^2 \\ &= T_1 - 2T_2 + T_3 \end{aligned} \quad (3)$$

$$\begin{aligned} T_1 &= \sum_{x \in X} \sum_{i=1}^d x_i^2 = \sum_{x \in X} \sum_{i=1}^d \left(\sum_{j=b-1}^0 2^j \cdot x_{ij} \right)^2 \\ &= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot \sum_{x \in X} x_{ij} \cdot x_{ik} \right) \end{aligned} \quad (4)$$

Let PX be a P-tree mask of set X that can quickly identify data points in X . PX is a bit pattern containing 1s and 0s, where bit 1 indicates that a point at that bit position belongs to X , while 0 indicates otherwise. Using this mask, (3) can be simplified by substituting:

$$\sum_{x \in X} x_{ij} \cdot x_{ik} \text{ with } COUNT(PX \wedge P_{ij} \wedge P_{ik}), \quad (5)$$

where COUNT is the aggregate operation that counts bit 1 in a given pattern and P_{ij} is the P-tree at i^{th} dimension and j^{th} bit position. Thus, T_1 can be simplified to be:

$$T_1 = \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(PX \wedge P_{ij} \wedge P_{ik}) \right) \quad (6)$$

Similarly for terms T_2 and T_3 , we get:

$$\begin{aligned} T_2 &= \sum_{x \in X} \sum_{i=1}^d x_i a_i \\ &= \sum_{i=1}^d a_i \left(\sum_{j=b-1}^0 2^j \cdot \sum_{x \in X} x_{ij} \right) \\ &= \sum_{i=1}^d a_i \sum_{j=b-1}^0 2^j \cdot COUNT(PX \wedge P_{ij}) \end{aligned} \quad (7)$$

$$T_3 = \sum_{x \in X} \sum_{i=1}^d a_i^2 = COUNT(PX) \cdot \sum_{i=1}^d a_i^2 \quad (8)$$

Therefore:

$$TV(X, a) = (X - a) \circ (X - a) \quad (9)$$

$$= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(PX \wedge P_{ij} \wedge P_{ik}) \right) -$$

$$2 \cdot \sum_{i=1}^d a_i \cdot \sum_{j=b-1}^0 2^j \cdot COUNT(PX \wedge P_{ij}) +$$

$$COUNT(PX) \cdot \sum_{i=1}^d a_i^2$$

Let us consider X as the relation R itself. In this case, the mask PX can be removed since all objects now belong to R . The formula then can be rewritten as:

$$\begin{aligned} TV(X, a) &= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(P_{ij} \wedge P_{ik}) \right) - \\ &2 \cdot \sum_{i=1}^d a_i \cdot \sum_{j=b-1}^0 2^j \cdot COUNT(P_{ij}) + |X| \sum_{i=1}^d a_i^2 \end{aligned} \quad (10)$$

where $|X|$ is the cardinality of R .

Note that the COUNT operations are independent from a (the input value). For classification problem where X is the training set (or the relation R itself), this independency allows us to pre-compute the count values in advance, retain them, and use them repeatedly when computing total variations. The reusability of count values expedites the computation of total variation significantly.

IV. ENHANCED SMART-TV WITH DIMENSIONAL PROJECTIONS PRUNING

SMART-TV classifier finds the candidates of neighbors by forming a total variation contour around the total variation of the unclassified object [1]. The objects within the contour are then considered as the superset of nearest neighbors. These objects are identified efficiently using P-tree range query algorithm without the need to scan the total variation values. The proposed algorithm further prunes the neighbors set by means of dimensional projections. After pruning, the k -nearest neighbors are searched from the set, and then, they vote to determine the class label of the unclassified object.

The algorithm consists of two phases: *preprocessing* and *classifying*. In the preprocessing phase, all processes are conducted only once, while in the classifying phase, the processes are repeated for every new unclassified object.

Let $R(A_1, \dots, A_n, C)$ be a training space and $X(A_1, \dots, A_n) = R[A_1, \dots, A_n]$ be the features subspace. $TV(X, a)$ is the total variation of X about a , and let $f(a) = TV(X, a) - TV(X, \mu)$.

Recall that:

$$\begin{aligned}
TV(X, a) &= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(P_{ij} \wedge P_{ik}) \right) - \\
& 2 \cdot \sum_{i=1}^d a_i \cdot \sum_{j=b-1}^0 2^j \cdot COUNT(P_{ij}) + |X| \sum_{i=1}^d a_i^2 \\
&= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(P_{ij} \wedge P_{ik}) \right) - \\
& 2 \cdot |X| \sum_{i=1}^d a_i \mu_i + |X| \sum_{i=1}^d a_i^2 \\
&= \sum_{i=1}^d \left(\sum_{j=b-1}^0 \sum_{k=b-1}^0 2^{j+k} \cdot COUNT(P_{ij} \wedge P_{ik}) \right) + \\
& |X| \left(-2 \cdot \sum_{i=1}^d a_i \mu_i + \sum_{i=1}^d a_i^2 \right)
\end{aligned}$$

Since $f(a) = TV(X, a) - TV(X, \mu)$, we get:

$$\begin{aligned}
&= |X| \left(-2 \cdot \sum_{i=1}^d a_i \mu_i + \sum_{i=1}^d a_i^2 \right) - \\
& |X| \left(-2 \cdot \sum_{i=1}^d \mu_i \mu_i + \sum_{i=1}^d \mu_i^2 \right) \\
&= |X| \left(-2 \cdot \sum_{i=1}^d (a_i \mu_i - \mu_i \mu_i) + \sum_{i=1}^d a_i^2 - \mu_i^2 \right) \\
&= |X| \left(\sum_{i=1}^d a_i^2 - 2 \sum_{i=1}^d \mu_i a_i + \sum_{i=1}^d \mu_i^2 \right) \\
&= |X| |a - \mu|^2 \tag{11}
\end{aligned}$$

In the preprocessing phase, the function $f(a)$ is applied to each training object, and derived P-trees of those functional values are created to incorporate a fast and efficient way to determine the candidates of neighbors. Since in large training sets the values of $f(a)$ can be very large and representing these large values in binary will require unnecessarily large number of bits, we let $g(a) = \ln(f(a)) = \ln|X| + \ln|a - \mu|^2$ to reduce the bit width.

By taking the first derivation of $g(a)$ fixing at d^{th} dimension, we find the gradient of g at a is $\frac{\partial g}{\partial a_d}(a) = \frac{2}{|a - \mu|^2} \cdot (a - \mu)_d$. The gradient is zero if and only if $a = \mu$, and the gradient length depends only on the length of vector $(a - \mu)$. This indicates that the isobars are hyper-circles. Note also that to avoid singularity when $a = \mu$, we add a constant 1 to the function $f(a)$ such that $g(a) = \ln(f(a) + 1)$. The shape of the graph is a funnel as

shown in Fig. 2.

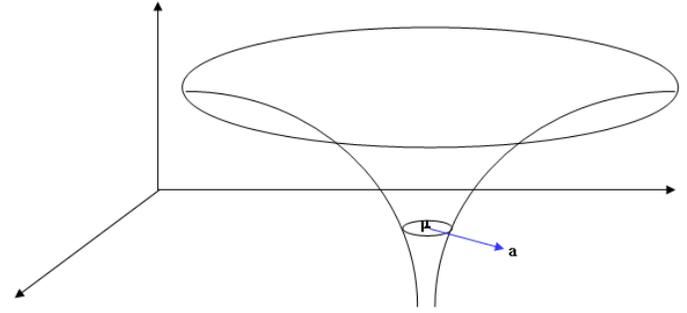


Fig. 2. The shape of graph $g(a)$.

A. Preprocessing Phase

In the preprocessing phase, we compute $g(x)$, $\forall x \in X$, and create derived P-trees of the functional values $g(x)$. The derived P-trees are stored together with the P-trees of the dataset. The complexity of this computation is $O(n)$ since the computation is applied to all objects in X . Furthermore, because the vector mean $\mu = (\mu_1, \mu_2, \dots, \mu_d)$ is used in the function, then the vector must be initially computed.

We compute the vector mean efficiently using P-tree vertical structure. An aggregate function COUNT is used to count the number of 1s in the bit pattern such that the sum of 1s in each vertical bit pattern is acquired first. The following formula shows how to compute the element of vector mean at i^{th} dimension vertically:

$$\begin{aligned}
\mu_i &= \frac{1}{|X|} \cdot \sum_{x \in X} x_i = \frac{1}{|X|} \cdot \sum_{x \in X} \sum_{j=b-1}^0 2^j \cdot x_{ij} = \\
& \frac{1}{|X|} \cdot \sum_{j=b-1}^0 2^j \cdot \sum_{x \in X} x_{ij} = \frac{1}{|X|} \cdot \sum_{j=b-1}^0 2^j \cdot COUNT(P_{ij}) \tag{12}
\end{aligned}$$

The complexity to compute the vector mean is $O(d \cdot b)$, where d is the number of dimensions and b is the bit width.

B. Classifying Phase

The following processes are performed on each unclassified object a . For simplicity, consider the objects in 2-dimensional space.

1. Determine vector $(a - \mu)$, where a is the new object and μ is the vector mean of the features space X .
2. Given an epsilon of the contour ($\epsilon > 0$), determine two vectors located in the lower and upper side of a by moving the ϵ unit inward toward μ along vector $(a - \mu)$ and moving the ϵ unit outward away from a . Let b and c be the two vectors in the lower and upper side of a respectively, then vector b and c can be determined using the following equations:

$$b = \mu + \left(1 - \frac{\varepsilon}{|a - \mu|}\right) \cdot (a - \mu) \quad (13)$$

$$c = \mu + \left(1 + \frac{\varepsilon}{|a - \mu|}\right) \cdot (a - \mu) \quad (14)$$

3. Calculate $g(b)$ and $g(c)$ such that $g(b) \leq g(a) \leq g(c)$ and determine the interval $[g(b), g(c)]$ that creates a contour over the functional line. The contour mask of the interval is created efficiently using the P-tree range query algorithm without having to scan the functional values one by one. The mask is a bit pattern containing 1s and 0s, where bit 1 indicates that the object is in the contour while 0 indicates otherwise. The objects within the pre-image of the contour in the original feature space are considered as the superset of neighbors (ε neighborhood of a or $\text{Nbrhd}(a, \varepsilon)$).

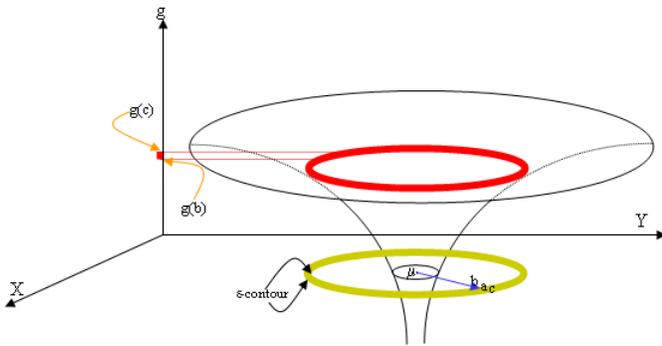


Fig. 3. The pre-image of the contour of interval $[g(b), g(c)]$ creates a $\text{Nbrhd}(a, \varepsilon)$.

4. Since the total variation contour is annular around the mean, the candidate set may contain neighbors that are actually far from the unclassified object a , e.g. located within the contour but in the opposite side of a . Therefore, an efficient pruning technique is required to eliminate the superfluous neighbors in the candidate set. We prune those neighbors using dimensional projections. For each dimension, a dimensional projection contour is created around the vector element of the unclassified object in that dimension. The size of the contour is specified by moving the ε unit away from the element of vector a in that dimension on both sides. It is important to note that because the training set was converted into P-trees vertical structure, thus, no additional derived P-trees are needed to perform the projection. The P-trees of each dimension can be used directly with no extra cost. Furthermore, a parameter ms (manageable size of candidate set) is needed for the pruning. This parameter specifies the upper bound of neighbors in the candidate set such that if the manageable size of neighbors is reached, the pruning process will be terminated, and the number of neighbors in the candidate set is considered small enough to be scanned to search for the k -nearest neighbors. The pruning

technique consists of two major steps. First, it obtains the count of neighbors in the pre-image of the total variation contour (candidate set) relative to a particular dimension. The rationale is to maintain the neighbors that are predominant (close to the unclassified object) in most dimensions so that, when the Euclidian distance is calculated (step 5 of the classifying phase), they are the true closest neighbors. The process of obtaining the count starts from the first dimension. The dimensional projection contour around the unclassified object a is formed, and the contour mask is created. Again, the mask is simply a bit pattern containing 1s and 0s, where bit 1 indicates that the object belongs to the candidate set when projected on that dimension while 0 indicates otherwise. The contour mask is then AND-ed with the mask of the pre-image of the total variation contour, and the total number of 1s is counted. Note that no neighbors are pruned at this point, only the count of 1s is obtained. The process continues for all dimensions, and at the end of the process, the counts are sorted in descending order. The second step of the pruning is to intersect each dimensional projection contour with the candidate set. The intersection starts based on the order of the count. The dimension with the highest count is intersected first, followed by the dimension with the second most count, and so forth. In each intersection, the number of neighbors in the candidate set is updated. From the implementation perspective, this intersection is simply a logical AND operation between the mask of the total variation contour and the mask of the dimensional projection contour. The second step of the pruning technique continues until a manageable size of neighbors is reached or all dimensional projection contours have been intersected with the candidate set. Fig. 4 illustrates the intersection between the pre-image of the total variation contour and the dimensional projection contours.

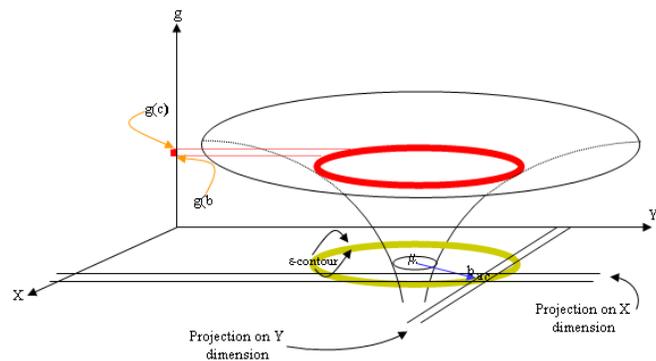


Fig. 4. Pruning the contour using dimensional projections.

5. Find the k -nearest neighbors from the remaining neighbors by calculating the Euclidian distance, $\forall x \in \text{Nbrhd}(a, \varepsilon)$,

$$d_2(x, a) = \sqrt{\sum_{i=1}^d (x_i - a_i)^2}$$

6. Let the k nearest neighbors vote using a weighted vote $w = e^{(-d*d)}$ to influence the vote of the k nearest neighbors. The weighting function is a Gaussian function that gives a nice drop off based on the distance of the neighbors to the unclassified object as shown in Fig. 5. The closer the neighbors the higher is the weight and vice versa.

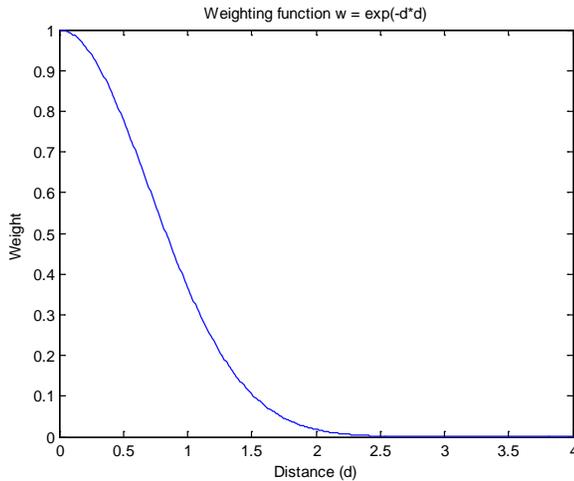


Fig. 5. Weighting function $W = e^{(-d*d)}$.

V. EXPERIMENTAL RESULTS

The experiments were conducted on an Intel Pentium 4 CPU 2.6 GHz machine with 3.8GB RAM, running Red Hat Linux version 2.4.20-8smp. We tested the proposed classifier using several datasets that have been previously used for classification problems, including those datasets from the Machine Learning Databases Repository at (<http://www.ics.uci.edu/~mllearn/MLRepository.html>). We also incorporated real-life datasets, such as Remote Sense Imaginary (RSI), OPTICS and Iris datasets, a few benchmark datasets used to evaluate classification algorithms.

The RSI dataset is a set of aerial photographs from the Best Management Plot (BMP) of Oakes Irrigation Test Area (OITA) near Oakes, North Dakota, taken in 1998. The images contain three bands: red, green, and blue. Each band has values in the range of 0 and 255, which in binary can be represented using 8 bits. The corresponding synchronized data for soil moisture, soil nitrate, and crop yield were also used, and the crop yield was selected as the class attribute. Combining all the bands and synchronized data, a dataset with 6 dimensions (five feature attributes and one class attribute) was obtained. To simulate different classes, the crop yield was divided into four different categories: *low yield* having intensity between 0 and 63, *medium low yield* having intensity between 64 and 127, *medium high yield* having intensity between 128 and 191, and *high yield* having intensity between 192 and 255. Three synthetic datasets were created to study the scalability and running time of the proposed classifier. The

cardinality of these datasets is varied from 32 to 96 million.

The KDDCUP 1999 dataset is the network intrusion dataset used in KDDCUP 1999 [10]. The dataset contains more than 4.8 million samples from the TCP dump. Each sample identifies a type of network intrusion such as *Normal*, *IP Sweep*, *Neptune*, *Port Sweep*, *Satan*, and *Smurf*.

The Wisconsin Diagnostic Breast Cancer (WDBC) dataset contains 569 diagnosed breast cancers with 30 real-valued features. The task is to predict two types of diagnoses as either *Benign* or *Malignant*.

The OPTICS dataset has eight different classes, containing 8,000 points in 2-dimensional space. The Iris dataset is the Iris plants dataset, created by R.A. Fisher [6]. The task is to classify Iris plants into one of the three varieties: *Iris Setosa*, *Iris Versicolor*, and *Iris Virginica*. Iris dataset contains 150 samples in a 4-dimensional space.

A. Speed and Scalability Comparison

We compared the performance in terms of speed using only RSI datasets. As mentioned previously, the cardinality of these datasets varies from 32 to 96 million. In this experiment, we compared the classification time of enhanced SMART-TV (SMART-TV with P-tree) against the SMART-TV with scan, P-KNN with HOBBIT metric, and a brute-force KNN. Note that SMART-TV with scan is the original SMART-TV [1] and it is slightly difference from the enhanced SMART-TV. SMART-TV with scan does not have derived P-trees of the total variation values of the training set and it determines the superset of neighbors in the total variation contour by scanning the total variation values of the training objects one by one. While scanning, the relative indexes of the objects in the contour are stored in an array. In this experiment, the same pruning technique was also applied to the original SMART-TV after the scanning process is completed. The candidate mask of the neighbors is created using P-Tree vertical structure API, and the parameter passed to the API is the array that holds the indexes of objects within the total variation contour. The results show that even though the scan was conducted on a single dimension containing the total variation values of the training objects, the original SMART-TV takes more time when compared to the enhanced SMART-TV that uses P-tree range query to determine the objects in the contour. Fig. 6 and 7 depict the classification time and scalability comparison of the algorithms. The speed was measured by observing the time required by CPU to complete the classification process. We learned that the enhanced SMART-TV and P-KNN are very comparable in terms of speed. Both algorithms are faster than the other two algorithms. For example, P-KNN takes 12.37 seconds on an average to classify using a dataset of size 96 million, while the enhanced SMART-TV takes 17.42 seconds. For the same dataset, SMART-TV with scan takes about 106.76 seconds to classify and KNN takes 891.58 seconds. KNN with sequential scan is almost three orders of magnitude slower than the enhanced SMART-TV and P-KNN algorithms. From this observation, it is clear that when the cardinality increases, the

classification time of KNN increases linearly. The algorithms that use P-tree vertical data structure are much faster. SMART-TV with scan is faster than KNN because it only scans a single dimension and compares the functional values of the unclassified object with the functional values of the training objects. In addition, the pruning technique is also incorporated in SMART-TV with scan, where as KNN has to scan and compute the Euclidian distance at the same time. However, the SMART-TV with scan is still slower than the enhanced SMART-TV and P-KNN.

The enhanced SMART-TV devotes much of its time for preprocessing as discussed previously in *Preprocessing Phase* section. The complexity to create P-trees and derived P-trees of the functional values is $O(n)$ since the computation is applied to all objects in the training set. The enhanced SMART-TV also needs to compute a vector mean. The complexity to compute the mean is $O(d \cdot b)$, where d is the number of dimensions and b is the bit width of the P-trees. In classification phase, the time is used to prune the superfluous neighbors in the candidate set. However, since the pruning is done using P-tree vertical masks and AND bit operation, the process is very fast. This is also true for the other vertical algorithm such as P-KNN. KNN classifier, on the other hand, does not build a model or structure in advance. It invests all the efforts in the classification phase when new objects arrive. Much of its time is used to calculate the distance between the training and the unclassified objects. Therefore, the computational complexity for KNN to classify each new unclassified sample is $O(n)$.

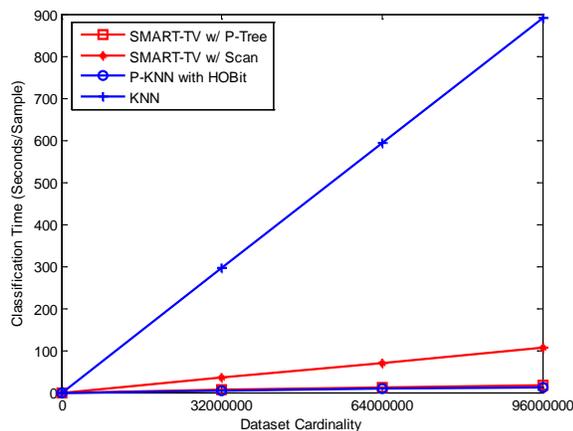


Fig. 6. Trend of classification time on RSI dataset.

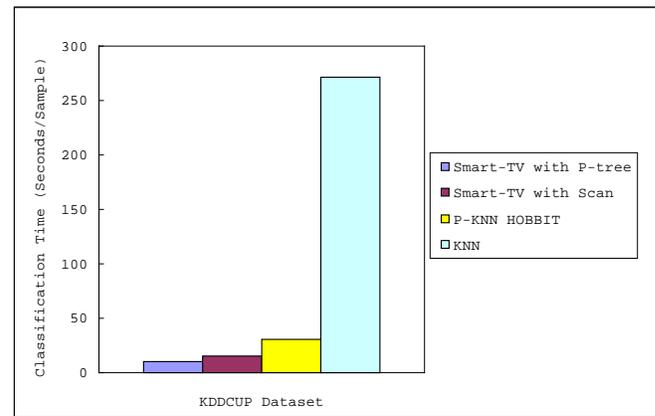


Fig. 7. Classification time on KDDCUP dataset.

B. Classification Accuracy Comparison

We examined the classification accuracy (F-score) using the experimental datasets with 5-folded cross validation model. We used $\epsilon = 0.1$ and $ms = 200$, $k=3$, $k=5$, and $k=7$ and compared the enhanced SMART-TV with P-KNN using HOBBIT, P-KNN using Equal Interval Neighborhood (EIN-ring), and KNN with linear search. The results are summarized in Table 1. From the results, it is clear that the classification accuracy of enhanced SMART-TV is very comparable to that of the KNN algorithm for all datasets. The performance of P-KNN using HOBBIT degrades significantly when classifying the dataset with high dimensions as shown in WDBC dataset.

TABLE 1
CLASSIFICATION ACCURACY COMPARISON ON VARIOUS DATASETS

Dataset	Classification Accuracy (F-Score)			
	Enhanced SMART-TV	P-KNN with HOBBIT	P-KNN with EIN	KNN
KDDCUP	0.93	0.91	n/a	0.93
OPTICS	0.99	0.99	0.99	0.99
IRIS	0.97	0.89	0.96	0.96
WDBC	0.94	0.39	0.96	0.97

VI. CONCLUSION

In this paper, we have presented an enhancement of SMART-TV classifier by introducing dimensional projections pruning. The evaluations show that the dimensional projections can effectively prune the superfluous neighbors that are distance away from the unclassified object, even though in terms of absolute difference of total variation, their values can be small. In terms of speed and scalability, the enhanced SMART-TV algorithm is very fast and comparable to the other vertical nearest neighbor algorithms, and in terms of accuracy, the enhanced SMART-TV is very comparable to that of KNN classifier.

REFERENCES

- [1] Abidin, T., & Perrizo, W. (2006). SMART-TV: A fast and scalable nearest neighbor based classifier for data mining, *Proceedings of the 21st Annual ACM Symposium on Applied Computing* (pp. 536-540). Dixon, France: Printing House, Inc.
- [2] Abidin, T., Perera, A., Serazi, M., & Perrizo, W. (2006). A vertical approach to computing set squared distance. *International Journal of Computers and their Applications (IJCA)*, 13 (2), 94-102.
- [3] Abidin, T., Dong, A., Li, H., & Perrizo, W. (2006). Efficient image classification on vertically decomposed data. *Proceedings of the 1st IEEE International Workshop on Multimedia Databases and Data Management*. Atlanta, Georgia: IEEE Computer Society.
- [4] Ankerst, M., Breunig, M., Kriegel, H-P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. *Proceeding of the ACM SIGMOD* (pp. 49-60). Philadelphia, P.A.
- [5] Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transaction on Information Theory*, 13, 21-27.
- [6] Demsar, J., Zupan, B., & Leban, G., (2004). Orange: from experimental machine learning to interactive data mining. White paper (www.ailab.si/orange). Faculty of Computer and Information Science, University of Ljubljana.
- [7] Ding, Q., Khan, M., Roy, A., & Perrizo, W. (2002). The p-tree algebra. *Proceedings of the 17th Annual ACM Symposium on Applied Computing* (pp. 426-431). Madrid, Spain.
- [8] Gray, J. (2004). The next database revolution. *Proceedings of the 10th ACM SIGMOD* (pp. 1-4). Paris.
- [9] Grother, P.J., Candela, G.T., & Blue, J.L. (1997). Fast implementations of nearest neighbor classifiers. *Pattern Recognition*, 30, 459-465.
- [10] Hettich, S., & Bay, S. D. (1999). The UCI KDD archive <http://kdd.ics.uci.edu>. Irvine, University of California, Department of Information and Computer Science.
- [11] Khan, M., Ding, Q., & Perrizo, W. (2002). K-nearest neighbor classification on spatial data stream using p-trees. *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 517-528). Taipei, Taiwan.
- [12] Perera, A., Abidin, T., Serazi, M., Hamer, G., & Perrizo, W. (2005). Vertical set squared distance based clustering without prior knowledge of k. *Proceedings of the 14th International Conference on Intelligent and Adaptive Systems and Software Engineering* (pp. 72-77). Toronto, Canada.
- [13] Perrizo, W. (2001). Peano count tree technology lab notes (Tech. Rep. NDSU-CS-TR-01-1). Computer Science Department, North Dakota State University.
- [14] Rahal, I., Ren, D., & Perrizo, W. (2004). A scalable vertical model for mining association rules. *Journal of Information & Knowledge Management (JIKM)*, 3(4), 317-329.
- [15] Ren, D., Wang, B., & Perrizo, W. (2004). RDF: a density-based outlier detection method using vertical data representation. *Proceedings of the 4th IEEE International Conference on Data Mining* (pp. 503-506).
- [16] Vries, A.P., Mamoulis, N., Nes, N., & Kersten, M. (2002). Efficient k-NN search on vertically decomposed data. *Proceedings of the ACM SIGMOD* (pp. 322-333). Madison, Wisconsin, USA.



Taufik F. Abidin received his B.Sc from Sepuluh Nopember Institute of Technology, Indonesia in 1993. Upon completing his B.Sc degree, he joined Syiah Kuala University in Banda Aceh, Indonesia. He received his M.Tech degree in Computer Science from RMIT University, Australia in 2000, and completed his Ph.D. in Computer Science at North Dakota State University (NDSU), USA in 2006. He received the ND EPSCoR Doctoral Dissertation Award from NDSU in 2005. He has been a Senior Software Engineer at Ask.com in New Jersey to develop algorithms and implement efficient production-level programs to improve web search results. His research interests include Data Mining, Database Systems, and Information Retrieval.



William Perrizo is a Professor of Computer Science at North Dakota State University. He holds a Ph.D. degree from the University of Minnesota, a Master's degree from the University of Wisconsin and a Bachelor's degree from St. John's University. He has been a Research Scientist at the IBM Advanced Business Systems Division and the U.S. Air Force Electronic Systems Division. His areas of expertise are Data Mining, Knowledge Discovery, Database Systems, Distributed Database Systems, High Speed Computer and Communications Networks, Precision Agriculture and Bioinformatics. He is a member of ISCA, ACM, IEEE, IAAA, and AAAS.