PINE - Podium Incremental Neighbor Evaluator for Classifying Spatial Data *

William Perrizo[1], Qin Ding[2], Anne Denton[1], Kirk Scott[3], Qiang Ding[1], Maleq Khan[4]

[1] Department of Computer Science, North Dakota State University, Fargo, ND 58102
{william.perrizo, anne.denton, qiang.ding}@ndsu.nodak.edu

[2] Department of Computer Science, Penn State Harrisburg, Middletown, PA 17057
qding@psu.edu

[3] Department of Mathematical Sciences, University of Alaska Anchorage, Anchorage, AK 99508
afkas@uaa.alaska.edu

[4] Department of Computer Science, Purdue University, West Lafayette, IN 47907
maleq@purdue.edu

ABSTRACT

Given a set of training data, nearest neighbor classification predicts the class value for an unknown tuple X by searching the training set for the k nearest neighbors to X and then classifying X according to the most frequent class among the k neighbors. Each of the k nearest neighbors casts an equal vote for the class of X. In this paper, we propose a new algorithm, Podium Incremental Neighbor Evaluator (PINE), in which nearest neighbors are weighted for voting. A metric called HOBBit is used as the distance metric, and a data structure, the P-tree*, is used for efficient implementation of the PINE algorithm on spatial data. Our experiments show that by using a Gaussian podium function, PINE outperforms the k-nearest neighbor (KNN) method in terms of classification accuracy for spatial data. In addition, in the PINE algorithm, all the instances are potential neighbors so that the value of k need not be pre-specified as in KNN methods. By assigning high weights to the nearest neighbors and low (even zero) weights to other neighbors, high classification accuracy can be achieved.

Keywords

Nearest neighbors, classification, data mining, spatial data

1. INTRODUCTION

Nearest neighbor classification is a lazy classifier. Given a set of training data, a k-nearest neighbor classifier predicts the class value for an unknown tuple X by searching the training set for the k nearest neighbors to X and then assigning to X the most common class among its k nearest neighbors.

In classical k-nearest neighbor (KNN) methods, each of the k nearest neighbors casts an equal vote for the class of X. We suggest that the accuracy can be increased by weighting the vote of different neighbors. Based on this, we propose an algorithm, called Podium Incremental Neighbor Evaluator (PINE), to achieve high accuracy by applying a podium function on the neighbors.

The idea of distance weighting is not new. For example, the concept of a "radial basis function" [7] is related to the idea of a podium function. However, to the best of our knowledge, applying the podium function to nearest neighbor classification is new.

Unlike other nearest neighbor classifiers, in PINE, no sub-sampling is done and no limit is placed on the number of neighbors, as in classical k-nearest neighbor classification techniques. The *podium* or distance weighting function (which can be user parameterized) establishes a *riser* height for each step of the podium weighting function as the distance from the sample grows. This approach gives users maximum flexibility in choosing just the right level of influence for each training sample in the entire training set.

Different metrics can be defined for "closeness" of two data points. In this paper, we use a metric, called HOBBit (High Order Basic Bit similarity), for spatial data. In addition, we use a data structure, the Peano Count Tree (P-tree), for efficient discovery of nearest neighbors, without scanning the database. P-trees [1,2,3] are a data mining-ready representation of integer-valued data. Count information is maintained to quickly perform data mining operations. P-trees represent bit information that is obtained from the data through a separation into bit planes. Their multi-level structure is chosen so as to achieve high compression. A consistent multi-level structure is maintained across all bit planes of all attributes. This is done so that a simple multi-way logical AND operation can be used to reconstruct count information for any attribute value or tuple.

The rest of the paper is organized as follows. In Section 2, we review the P-tree structure. In Section 3, we introduce the

---

* Patents are pending on the P-tree technology. This work is partially supported by GSA Grant ACT# K96130308.

SAC 2003, Melbourne, Florida, USA

HOBBit metric and detail our PINE algorithm. Performance analysis is given in Section 4. Section 5 concludes the paper.

2. THE P-TREE STRUCTURE REVIEW

We use a structure, called a Peano Count Tree (P-tree), to represent the spatial data. We first split each attribute value into bits. For example, for image data, each band is an attribute, and the value is represented as a byte (8 bits). The file for each individual bit is called a bSQ file. For an attribute with m-bit values, we have m bSQ files. We organize each bSQ bit file, Bij (the file constructed from the j^{th} bits of i^{th} attribute), into a P-tree. Let's look at an example in Figure 1.

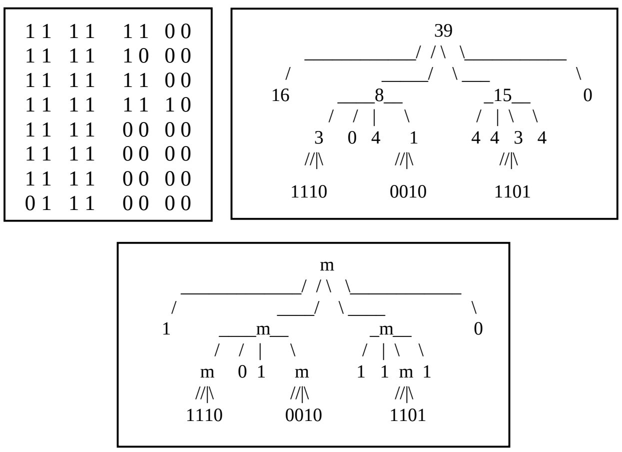

Figure 1. P-tree and PM-tree

In this example, 39 is the count of 1's in the entire image, called root count. The numbers at the next level, 16, 8, 15 and 16, are the 1-bit counts for the four major quadrants. Since the first and last quadrant is made up of entirely 1-bits and 0-bits respectively (called pure1 and pure0 quadrant respectively), we do not need sub-trees for these two quadrants. This pattern is continued recursively. Recursive raster ordering is called the Peano or Z-ordering in the literature – therefore, the name Peano Count trees. The process will definitely terminate at the "leaf" level where each quadrant is a 1-row-1-column quadrant.

For each band, assuming 8-bit data values, we get 8 basic P-trees, one for each bit position. For band Bi we will label the basic P-trees, $P_{i,1}$, $P_{i,2}$, …, $P_{i,8}$, where $P_{i,j}$ is a lossless representation of the j^{th} bit of the values from the i^{th} band. However, the $P_{i,j}$ provides much more information and is structured to facilitate many important data mining processes.

For efficient implementation, we use a variation of P-trees, called PM-tree (Pure Mask tree), in which mask instead of count is used. In the PM-tree, 3-value logic is used, i.e, 11 represents a pure1 quadrant, 00 represents a pure0 quadrant and 01 represents a mixed quadrant. To simplify, we use 1 for pure1, 0 for pure0, and m for mixed. This is illustrated in the 3^{rd} part of Figure 1.

P-tree algebra contains operators, AND, OR, NOT and XOR, which are the pixel-by-pixel logical operations on P-trees [3]. The NOT operation is a straightforward translation of each count to its quadrant-complement. The AND and OR operations are shown in Figure 2.

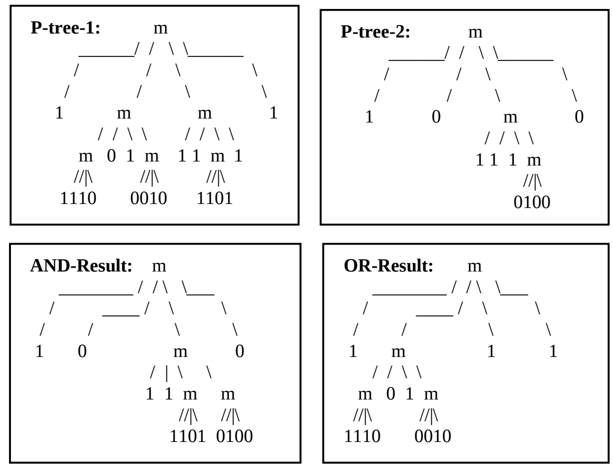

Figure 2. P-tree Algebra (AND and OR)

The basic P-trees can be combined using simple logical operations to produce P-trees for the original values (at any level of precision, 1-bit precision, 2-bit precision, etc.). We let $P_{b,v}$ denote the P-tree for attribute, b, and value, v, where v can be expressed in any bit precision. Using the 8-bit precision for values, $P_{b,11010011}$ can be constructed from the basic P-trees as:

$$P_{b,11010011} = P_{b1} \text{ AND } P_{b2} \text{ AND } P_{b3}' \text{ AND } P_{b4} \text{ AND } P_{b5}' \text{ AND } P_{b6}' \text{ AND } P_{b7} \text{ AND } P_{b8}.$$

Where ′ indicates the NOT operation. The AND operation is simply the pixel-wise AND of the bits.

Similarly, the data in the relational format can be represented as P-trees also. For any combination of values, $(v_1,v_2,…,v_n)$, where v_i is from attribute-i, the quadrant-wise count of occurrences of this combination of values is given by:

$$P_{(v1,v2,…,vn)} = P_{1,v1} \text{ AND } P_{2,v2} \text{ AND } … \text{ AND } P_{n,vn}$$

P-trees are implemented in an efficient way which facilitate fast P-tree operations and considerable saving of space. More details can be found in [3, 9].

3. DISTANCE-WEIGHTED (PODIUM) NEIGHBOR CLASSIFICATION USING P-TREES

In classical k-nearest neighbor classification techniques, there is a limit placed on the number of neighbors. In our distance-weighted neighbor classification approach, the *podium* or distance weighting function (which can be user parameterized) establishes a *riser* height for each step of the podium weighting function as the distance from the sample grows. This approach gives users maximum flexibility in choosing just the right level of influence for each training sample in the entire training set. The real question is, can this level of flexibility be offered without imposing a severe penalty with respect to the speed of the classifier. Traditionally, sub-sampling, neighbor-limiting and other restrictions are introduced precisely to ensure that the algorithm will finish its classification in reasonable time (or at all!). The use of the compressed, data-mining-ready data structure, the P-tree, in fact, makes PINE even faster than traditional methods. This is critically important in classification since data are typically never discarded and therefore the training set will grow without bound. The classification technique must scale well or it will quickly become unusable in this setting. PINE scales well since its accuracy increases as the training set grows while its speed remains very reasonable (see the performance study below). Furthermore, since PINE is *lazy* (does not require a training phase in which a closed form classifier is pre-built), it does not incur the expensive delays required for rebuilding a classifier when new training data arrives. Thus, PINE gives us a faster and more accurate classifier.

Before explaining how distance-weighted neighbor classification (Podium Incremental Neighbor Evaluator or PINE) using P-trees works, we give an overview of the technique of *k*-nearest-neighbor classification. In *k*-nearest neighbor (KNN) the basic idea is that the tuples that most likely belong to the same class are those that are similar in the other attributes. This continuity assumption is consistent with the properties of a spatial neighborhood.

Based on some pre-selected distance metric or similarity measure, such as Euclidean distance, classical KNN finds the *k* most similar or nearest training samples to an unclassified sample and assigns the plurality class of those *k* samples to the new sample [5, 6]. The value for *k* is pre-selected by the user based on the accuracy required (usually the larger the value of *k*, the more accurate the classifier) and the delay time required for classifying with that k-value (usually the larger the value of k the slower the classifier). The steps of the classification process are:

1) Determine a suitable distance metric.
2) Find the *k* nearest neighbors (NNs) using the selected distance metric.
3) Find the plurality class of the *k*-nearest neighbors (voting on the class labels of the NNs).
4) Assign that class to the sample to be classified.

We use a new *HOBBit* distance (see next section), which provides an efficient method of computation based on P-trees. Instead of examining individual training samples to find the nearest neighbors, we start our initial *neighborhood* of the target sample within a specified distance in the feature space based on this metric, and then successively expand the neighborhood area until there are at least *k* tuples in the neighborhood set.

Of course, there may be more boundary neighbors equidistant from the sample than are necessary to complete the *k* nearest neighbor set, in which case, one can either use the larger set or arbitrarily ignore some of them. Other methods find the exact k nearest neighbor set, since that is easiest using traditional techniques although it is clear that allowing some samples at that distance to vote and not others, will skew the result. Instead, the P-tree-based KNN approach [4] builds a *closed* nearest neighbor set (closed-KNN), that is, we include all of the boundary neighbors. The inductive definition of the closed-KNN set is given below.

(a) If $x \in$ KNN, then $x \in$ closed-KNN
(b) If $x \in$ closed-KNN and $d(T,y) \le d(T,x)$, then $y \in$ closed-KNN, where, $d(T,x)$ is the distance of x from target T.
(c) Closed-KNN does not contain any tuple which cannot be produced by steps a and b.

Experimental results [4] show closed-KNN yields higher classification accuracy than KNN does. If there are many tuples on the boundary, inclusion of some but not all of them skews the voting mechanism. The P-tree implementation requires no extra computation to find the closed-KNN. Our neighborhood expansion mechanism automatically includes the entire boundary of the neighborhood. P-tree algorithms avoid the examination of individual data points, which improves the classification efficiency.

3.1 Higher Order Basic bit (HOBBit) distance

For two data points, X = <x_1, x_2, x_3, …, x_{n-1}> and Y = <y_1, y_2, y_3, …, y_{n-1}>, the **Euclidean** similarity function is defined as $d_2(X,Y) = \sqrt{\sum_{i=1}^{n-1} (x_i - y_i)^2}$. It can be generalized to the **Minkowski** similarity function, $d_q(X,Y) = \sqrt[q]{\sum_{i=1}^{n-1} w_i |x_i - y_i|^q}$. If q = 2, this gives the Euclidean function. If q = 1, it gives the **Manhattan** distance, which is $d_1(X,Y) = \sum_{i=1}^{n-1} |x_i - y_i|$. If q = ∝, it gives the **max** function $d_\infty(X,Y) = \max_{i=1}^{n-1} |x_i - y_i|$.

The HOBBit [4] metric measures distance based on the most significant consecutive bit positions starting from the left (the highest order bit). Similarity or closeness is of interest. When comparing two values bitwise from left to right, once a difference is found, any further comparisons are not needed.

The HOBBit similarity between two integers *A* and *B* is defined by

$$S_H(A, B) = \max\{s \mid 0 \le i \le s \Rightarrow a_i = b_i\} \quad \dots \text{(eq. 1)}$$

where a_i and b_i are the i^{th} bits of *A* and *B* respectively.

The HOBBit distance between two tuples *X* and *Y* is defined by

$$d_H(X,Y) = \max_{i=1}^{n-1}\{m - S_H(x_i,y_i)\} \quad \dots \text{(eq. 2)}$$

where *m* is the number of bits in binary representations of the values; *n - 1* is the number of attributes used for measuring distance (the n^{th} being the class attribute); and x_i and y_i are the i^{th} attributes of tuples *X* and *Y*. The HOBBit distance between two tuples is a function of the least similar pairing of attribute values in them.

From the experiments, we found that HOBBit distance is more natural for spatial data than other distance metrics.

3.2 Closed-KNN using P-trees

To find the closed KNN set, first we look for the tuples which are identical to the target tuple in all 8 bits of all bands, i.e. the tuples, X, having distance from the target T, $d_H(X,T) = 0$. If, for instance, t_1=105 ($01101001_b = 105_d$) is a target attribute value, the initial interval of interest is [105, 105] ([01101001, 01101001]). If over all tuples the number of matches is less than k, we compare attributes on the basis of the 7 most significant bits, not caring about the 8^{th} bit. The expanded interval of interest would be [104,105] ([01101000, 01101001] or [0110100-, 0110100-]). If k matches still haven't been found, removing one more bit from the right gives the interval [104, 107] ([011010--, 011010--]). Continuing to remove bits from the right we get intervals, [104, 111], then [96, 111] and so on.

This process is implemented using P-trees as follows. $P_{i,j}$ is the basic P-tree for bit j of band i and $P'_{i,j}$ is the complement of $P_{i,j}$. Let, $b_{i,j}$ be the j^{th} bit of the i^{th} band of the target tuple, and for implementation purposes let the representation of the P-tree depend on the value of b_{ij}. Define:

$$Pt_{i,j} = P_{i,j}, \quad \text{if } b_{i,j} = 1,$$
$$\quad = P'_{i,j}, \quad \text{otherwise.}$$

Then the root count of $Pt_{i,j}$ is the number of tuples in the training dataset having the same value as the j^{th} bit of the i^{th} band of the target tuple. Define:

$$Pv_{i,1\text{-}j} = Pt_{i,1} \,\&\, Pt_{i,2} \,\&\, Pt_{i,3} \,\&\, \ldots \,\&\, Pt_{i,j}, \qquad \ldots \text{(eq. 3)}$$

where & is the P-tree AND operator and n is the number of bands. $Pv_{i,1\text{-}j}$ counts the tuples having the same bit values as the target tuple in the higher order j bits of i^{th} band. Then a neighborhood P-tree can be formed as follows:

$$Pnn(j) = Pv_{1,1\text{-}j} \,\&\, Pv_{2,1\text{-}j} \,\&\, Pv_{3,1\text{-}j} \,\&\, \ldots \,\&\, Pv_{n\text{-}1,1\text{-}j} \quad \ldots \text{(eq. 4)}$$

We calculate the initial neighborhood P-tree, *Pnn*(8), matching exactly in all bands, considering 8-bit values. Then we calculate *Pnn*(7), matching in 7 higher order bits; then Then *Pnn*(6) and so on. We continue as long as the root count of *Pnn*(j) is less than k. Let us denote the final *Pnn*(j) by *Pcnn*. *Pcnn* represents the closed-KNN set and the root count of *Pcnn* is the number of the nearest tuples. A 1 bit in *Pcnn* for a tuple means that the tuple is in the closed-KNN set. For the purpose of classification, we don't need to consider all bits in the class band. If the class band is 8 bits long, there are 256 possible classes. Instead, we partition the class band values into fewer, say 8, groups by truncating the 5 least significant bits. The 8 classes are 0, 1, 2, …, 7. Using the leftmost 3 bits we construct the value P-trees $P_n(0)$, $P_n(1)$, …, $P_n(7)$. The P-tree *Pcnn* & $P_n(i)$ represents the tuples having a class value i that are in the closed-KNN set, *Pcnn*. An i, which yields the maximum root count of *Pcnn* & $P_n(i)$ is the plurality class; that is

$$\text{predicted class} = \arg\max_i \{\mathrm{RC}(Pcnn \,\&\, P_n(i))\} \quad \ldots \text{(eq. 5)}$$

where, RC(P) is the root count of P. More details about closed-KNN can be found in [4].

3.3 The Distance Weighted Nearest Neighbor (Podium Incremental Neighbor Evaluator or PINE) Method using P-trees

The continuity assumption of KNN tells us that tuples that are more similar to a given tuple have more influence on classification than tuples that are less similar. Therefore giving more voting weight to closer tuples than distant tuples increases the classification accuracy. Instead of considering the k nearest neighbors, we include all of the points, using the largest weight, 1, for those matching exactly, and the smallest weight, 0, for those furthest away. Many weighting functions which decreases with distance, can be used (e.g., Gaussian, Kriging, etc). Remaining consistent with the neighborhood rings (Figure 3) using the HOBBit distance, we can apply, for instance, a linear podium function (Figure 4), which decreases step-by-step with distance.

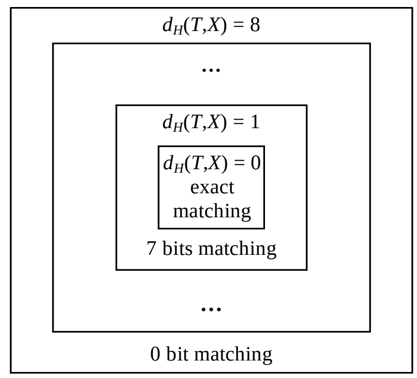

Figure 3. Neighborhood rings using HOBBit

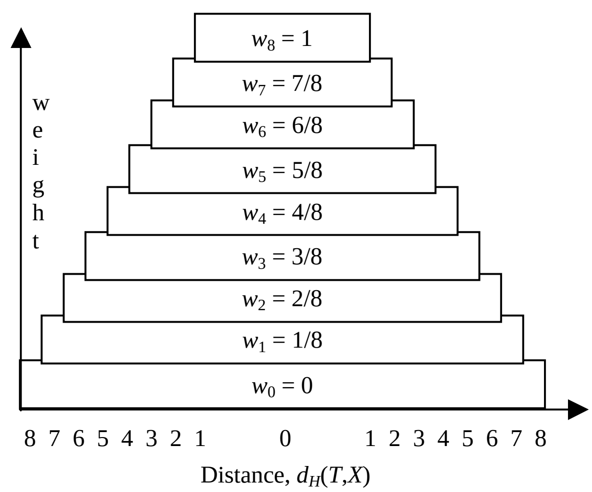

Figure 4. Linear podium function

Note that the HOBBit distance metric is ideally suited to the definition of neighborhood rings, because the range of points that are considered equidistant grows exponentially with distance from the center. Adjusting weights is particularly important for small to intermediate distances where the podiums are small. At larger distances where fine-tuning is less important the HOBBit distance remains unchanged over a large range, i.e., podiums are wider. Ideally, the 0-weighted ring should include all training samples that are judged to be too far away (by a domain expert) to influence class.

We number the rings from 0 (outermost) to m (innermost). Let w_j be the weight associated with the ring j. Let c_{ij} be the number of neighbor tuples in the ring j belonging to the class i. Then the total weight vote by the class i is given by:

$$V(i) = \sum_{j=0}^{m} w_j c_{ij} \qquad \text{… … … (eq. 6)}$$

This can easily be transformed to:

$$V(i) = w_0 \sum_{k=0}^{m} c_{il} + \sum_{j=1}^{m} \left\{ (w_j - w_{j-1}) \sum_{k=j}^{m} c_{ik} \right\} \qquad \text{… …… (eq. 7)}$$

Let circle j be the circle formed by the rings j, j+1, …, m, that is, the ring j including all of its inner rings. Referring to eq. 4, the P-tree, $Pnn(j)$, represents all of the tuples in the circle j. Therefore, $\{Pnn(j)\ \&\ P_n(i)\}$ represents the tuples in the circle j and class i; $P_n(i)$ is the P-tree for class i. Hence:

$$\sum_{k=j}^{m} c_{ik} = \mathrm{RC}\{Pnn(j)\ \&\ P_n(i)\}, \qquad \text{… … … (eq. 8)}$$

$$V(i) = w_0 \mathrm{RC}\{Pnn(0)\ \&\ P_n(i)\} + \sum_{j=1}^{m} \left[(w_j - w_{j-1}) \mathrm{RC}\{Pnn(j)\ \&\ P_n(i)\} \right]$$

… … … (eq. 9)

An i which yields the maximum weighted vote, $V(i)$, is the plurality class or the predicted class; that is:

$$\text{predicted class} = \arg\max_i \{V(i)\} \qquad \text{… … … (eq. 10)}$$

4. PERFORMANCE ANALYSIS

We have performed experiments to evaluate PINE on the real data sets including the aerial TIFF image (with Red, Green and Blue band reflectance values), moisture, nitrate, and yield map of the Oaks area in North Dakota. In these datasets yield is the class label attribute. The data sets are available at [8]. We formed test set and training set of equal size and tested KNN with Manhattan, Euclidean, Max, and HOBBit distance metrics; and closed-KNN with the HOBBit metric, and Podium Incremental Neighbor Evaluator (PINE). In PINE, HOBBit was used as the distance function and the Gaussian function was used as the podium function. We specify variance σ as 2^4, and the function is $\exp(-(2^{2*d}) / (2*\sigma^2))$, where d is the HOBBit distance. Therefore, the mapping is given in Table 1.

Table 1. Gaussian weighs as the function of HoBBit distance

HOBBit distance	0	1	2	3	4	5	6	7
Gaussian weigh	1.00	1.00	0.97	0.88	0.61	0.14	0.00	0.00

The accuracies of different implementations are given in Figure 5 for one dataset. We got quite similar results for other spatial datasets, which are consistent with our analysis about the properties of spatial data.

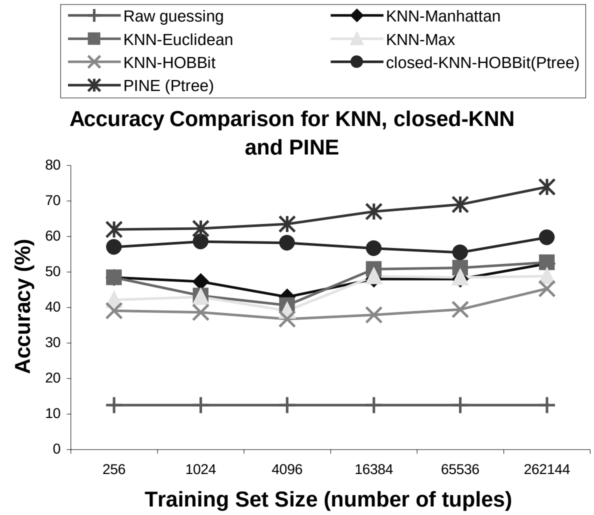

Figure 5. Accuracy comparison for KNN, closed-KNN and PINE using different metrics

We see that PINE performs better than closed-KNN as we expected. Especially when the training set size increases, the improvement of PINE over closed-KNN is more apparent. In our previous work [16], we have already explained the improvement of closed-KNN over KNN using various metrics. All these classifiers work well compared to raw guessing, which is 12.5% in this data set with 8 class values.

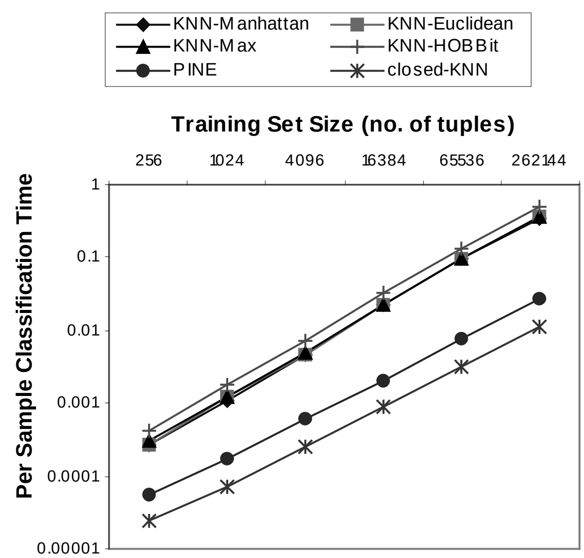

Figure 6. Classification time per sample (Size and classification time are plotted in logarithmic scale)

In terms of speed, from Figure 6, we see that there is some additional time cost of using PINE, however, this additional cost is relatively small. Notice that both size and classification time are plotted in logarithmic scale. We observe that both closed-KNN and PINE are much faster than KNN using any metric. On the average, PINE is eight times faster than the KNN, and closed-KNN is 10 times faster. Both PINE and closed-KNN increase at a lower rate than KNN methods do when the training set size increases.

5. CONCLUSIONS

In this paper, we propose a Podium Incremental Neighbor Evaluator (PINE), for classification on spatial data. We use the HOBBit metric and P-tree data structure for efficient implementation of PINE. Performance analysis shows that PINE outperforms KNN methods in terms of accuracy and speed of classification on spatial data.

PINE is particularly useful for classification on data streams. In data streams, new data arrives very quickly, so both speed and accuracy are important issues. Achieving high speed using P-trees, and high accuracy using the Podium Incremental Neighbor Evaluator (PINE) provides a classification method that is well suited to the classification of stream data. Besides spatial and stream data, this work has potential applications in other areas, such as DNA micro array data and medical image analysis.

REFERENCES

[1] William Perrizo, Qin Ding, Qiang Ding, Amalendu Roy, “On mining satellite and other Remotely Sensed Images”, in Proceedings of Workshop on Research Issues on Data Mining and Knowledge Discovery, 2001, p 33-44.

[2] William Perrizo, Peano Count Tree Technolgy, Tchnical Report NDSU-CSOR-TR-01-1, 2001.

[3] Qin Ding, Maleq Khan, Amalendu Roy, “The P-tree Algebra”, ACM Symposium on Applied Computing, 2002, pp. 426-431.

[4] Maleq Khan, Qin Ding, William Perrizo, “*k*-Nearest Neighbor Classification on Spatial Data Streams Using P-Trees”, PAKDD 2002, Springer-Verlag, LNAI 2336, 2002, pp. 517-528.

[5] Dasarathy, B.V., “Nearest-Neighbor Classification Techniques”, IEEE Computer Society Press, Los Alomitos, CA, 1991.

[6] M. James, “Classification Algorithms”, New York: John Wiley & Sons, 1985.

[7] M. A. Neifeld and D. Psaltis, “Optical Implementations of Radial Basis Classifiers”, Applied Optics, Vol. 32, No. 8, 1993, pp. 1370-1379.

[8] TIFF image data sets. Available at http://midas-10.cs.ndsu.nodak.edu/data/images/

[9] William Perrizo, Qin Ding, Qiang Ding and Amalendu Roy, “Deriving High Confidence Rules from Spatial Data using Peano Count Trees”, in Proceedings of International Conference on Web-Age Information Management (WAIM'01), Springer-Verlag, Lecture Notes in Computer Science 2118, July 2001, pp. 91-102.