

# Lazy Classifiers Using P-trees \*

William Perrizo<sup>1</sup>, Qin Ding<sup>2</sup>, Anne Denton<sup>1</sup>

<sup>1</sup> Department of Computer Science  
North Dakota State University  
Fargo, ND 58015  
{William.Perrizo, Anne.Denton}@ndsu.nodak.edu

<sup>2</sup> Department of Computer Science  
Penn State Harrisburg  
Middletown, PA 17057  
qding@psu.edu

## Abstract

Lazy classifiers store all of the training samples and do not build a classifier until a new sample needs to be classified. It differs from eager classifiers, such as decision tree induction, which build a general model (such as a decision tree) before receiving new samples. K-nearest neighbor (KNN) classification is a typical 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. Lazy classifiers are faster at training time than eager classifiers, but slower at predicating time since all computation is delayed to that time. In this paper, we introduce approaches to efficient construction of lazy classifiers, using a data structure, Peano Count Tree (P-tree)\*. P-tree is a lossless and compressed representation of the original data that records the count information to facilitate efficient data mining. With P-tree structure, we introduced two classifiers, P-tree based k-nearest neighbor classifier (PKNN), and Podium Incremental Neighbor Evaluator (PINE). Performance analysis shows that our algorithms outperform classical KNN methods.

## Keywords:

Lazy classifier, classification, nearest neighbor

## 1. INTRODUCTION

Classifiers can be divided into two categories, lazy classifiers and eager classifiers. In lazy classifiers, no general model is built until

a new sample needs to be classified. It differs from eager classifiers, such as decision tree induction, which build a general model (such as a decision tree) before receiving new samples.

K-nearest neighbor (KNN) classification is a typical 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.

Lazy classifiers are simple and effective. However, it's slow at predicating time since all computation is delayed to that time. In this paper, we introduce two lazy classifiers, P-tree based k-nearest neighbor classifier (PKNN), and Podium Incremental Neighbor Evaluator (PINE). We use a data structure, Peano Count Tree (P-tree), for efficient construction of those lazy classifiers. P-tree is a lossless and compressed representation of the original data that records the counts to facilitate efficient data mining.

The rest of the paper is organized as follows. Section 2 reviews the P-tree structure. Section 3 introduces two classifiers, PKNN and PINE. 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) [1, 2], 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,  $B_{ij}$  (the file constructed from the  $j^{\text{th}}$  bits of

---

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

$i^{\text{th}}$  attribute), into a P-tree. An example is given in Figure 1.

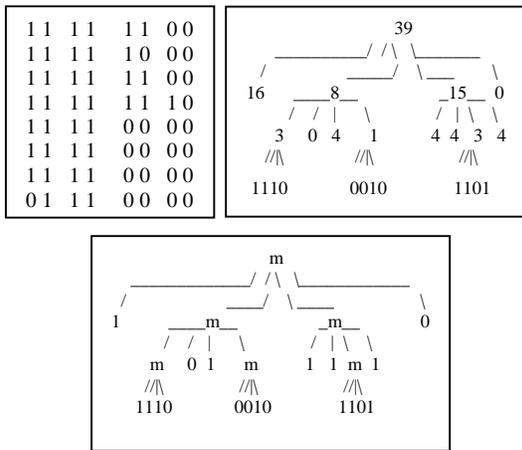


Figure 1. P-tree and PM-tree

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  $B_i$  we will label the basic P-trees,  $P_{i,1}, P_{i,2}, \dots, P_{i,8}$ , where  $P_{i,j}$  is a lossless representation of the  $j^{\text{th}}$  bit of the values from the  $i^{\text{th}}$  band. However, the  $P_{ij}$  provide much more information and are 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<sup>rd</sup> 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.

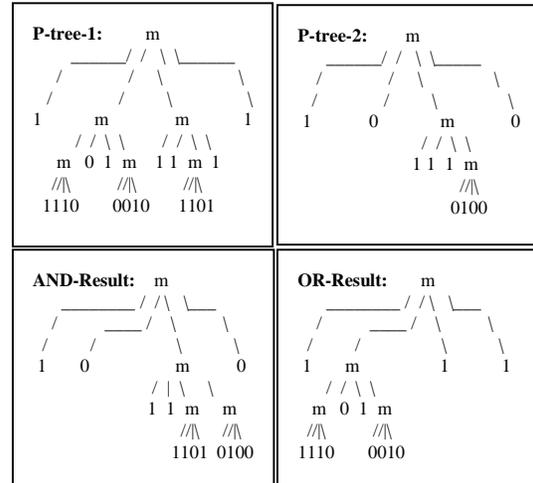


Figure 2. P-tree Algebra

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,  $(v_1, v_2, \dots, v_n)$ , where  $v_i$  is from attribute- $i$ , the quadrant-wise count of occurrences of this combination of values is given by:

$$P_{(v_1, v_2, \dots, v_n)} = P_{1, v_1} \text{ AND } P_{2, v_2} \text{ AND } \dots \text{ AND } P_{n, v_n}$$

### 3. LAZY CLASSIFIERS USING P-TREES

#### 3.1 Distance Metrics

The distance metric is required to determine the closeness of samples. Various distance

metrics have been proposed. For two data points,  $X = \langle x_1, x_2, x_3, \dots, x_{n-1} \rangle$  and  $Y = \langle y_1, y_2, y_3, \dots, y_{n-1} \rangle$ , 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  $q = \infty$ , it gives the

**max** function  $d_\infty(X, Y) = \max_{i=1}^{n-1} |x_i - y_i|$ .

We proposed a metric using P-trees, called HOBBit [3]. The HOBBit metric measures distance based on the most significant consecutive bit positions starting from the left (the highest order bit). The HOBBit similarity between two integers  $A$  and  $B$  is defined by

$S_H(A, B) = \max\{s / 0 \leq i \leq s \Rightarrow a_i = b_i\} \dots$  (eq. 1)  
where  $a_i$  and  $b_i$  are the  $i^{\text{th}}$  bits of  $A$  and  $B$  respectively.

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

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

where  $m$  is the number of bits in binary representations of the values;  $n$  is the number of attributes used for measuring distance; and  $x_i$  and  $y_i$  are the  $i^{\text{th}}$  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.

### 3.2 PKNN – K-nearest Neighbor Classifier using P-trees

The basic idea of k-nearest neighbor (KNN) classification is that similar tuples most likely belongs to the same class (a continuity assumption). Based on some pre-selected distance metric, it finds the k most similar or nearest training samples and assign the plurality class of those k samples to the new sample. It consists four steps:

1) Determine a suitable distance metric.

2) Find the k nearest neighbors using the selected distance metric.

3) Find the plurality class of the k-nearest neighbors.

4) Assign that class to the sample to be classified.

Database scans are performed to find the nearest neighbors, which is the bottleneck in the method. In PKNN, by using P-trees, we can quickly calculate the nearest neighbor for the given sample. For example, given a tuple  $\langle v_1, v_2, \dots, v_n \rangle$ , we can get the exact count of this tuple by calculating the root count of

$$P_{(v_1, v_2, \dots, v_n)} = P_{1, v_1} \text{ AND } P_{2, v_2} \text{ AND } \dots \text{ AND } P_{n, v_n}$$

With HOBBit metric, we first calculate all exact matches with the given data sample  $X$ , then extend to data points with distance 1 (with at most one bit difference in each attribute) from  $X$ , and so on, until we get k nearest neighbors.

Instead of just getting exact k nearest neighbors, we take the closure of the k-NN set, that is, we include all of the boundary neighbors that have the same distance as the kth neighbor to the sample point. Obviously closed-KNN is a superset of KNN set. As we pointed out later in the performance analysis, closed-KNN improves the accuracy obviously.

### 3.3 PINE – Podium Nearest Neighbor Classifier using P-trees

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 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).

#### 4. PERFORMANCE ANALYSIS

We have performed experiments on the real image data sets, which consist of aerial TIFF image (with Red, Green and Blue bands), 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 tested KNN with various metrics, PKNN, and PINE. The accuracies of these different implementations are given in Figure 3.

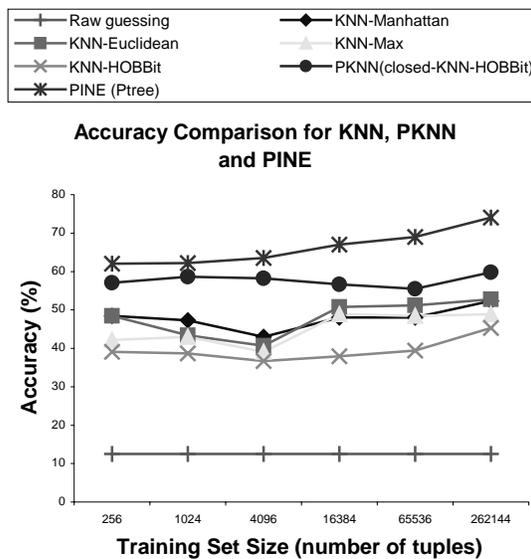


Figure 3. Accuracy comparison for KNN, PKNN and PINE

We see that PKNN performs better than KNN using various metrics, including Euclidean, Manhattan, Max and HOBBit. PINE further improves the accuracy. Especially when the training set size increases, the improvement is more apparent. Both PKNN and PINE work well compared to raw guessing, which is 12.5% in this data set with 8 class values.

#### 5. CONCLUSION

In this paper, we introduce two lazy classifiers using P-trees, PKNN and PINE. In PKNN, P-tree structure facilitates efficient computation of the nearest neighbors. The computation is done by performing fast P-tree

operations instead of scanning the database. In addition, with proposed closed-KNN, accuracy is increased. In PINE method, podium function is applied to different neighbors so accuracy is further improved.

Lazy classifiers are particularly useful for classification on data streams. In data streams, new data keep arriving, so building a new classifier each time can be very expensive. By using P-trees, we can build lazy classifiers efficiently and effectively for stream data.

#### 6. 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 Technology, Technical Report NDSU-CSOR-TR-01-1, 2001.
- [3] 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.
- [4] Dasarathy, B.V., "Nearest-Neighbor Classification Techniques", IEEE Computer Society Press, Los Alomitos, CA, 1991.
- [5] M. James, "Classification Algorithms", New York: John Wiley & Sons, 1985.
- [6] M. A. Neifeld and D. Psaltis, "Optical Implementations of Radial Basis Classifiers", Applied Optics, Vol. 32, No. 8, 1993, pp. 1370-1379.
- [7] TIFF image data sets. Available at <http://midas-10.cs.ndsu.nodak.edu/data/images/>
- [8] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.