Efficient Proximal Support Vector Machine for Spatial Data

Fei Pan, Baoying Wang, Dongmei Ren, Xin Hu, William Perrizo
Computer Science Department
North Dakota State University
Fargo, ND 58105, USA
{fei.pan, baoying.wang, dongmei.ren, xin.hu, william.perrizo} @ndsu.nodak.edu

Abstract

With more and more spatial data being collected data mining for spatial data has become a rapidly evolving research area. Support vector machine (SVM) as a powerful tool for data classification has the potentials in spatial data mining. However, traditional SVM involves solving a quadratic optimization problem that requires considerably long computation time for large data sets. In this paper, we propose a new efficient proximal Support Vector Machine (P-SVM) using Peano tree. Peano tree (P-tree) is a vertical compressed bitwise quadrant tree that is processed horizontally through logical operations. P-SVM exploits a unique neighborhood search method, i.e., EIN-ring based neighborhood search to find the boundary sentries. The final boundary hyperplanes of test data are determined by their \(d\)-nearest boundary sentries, which are calculated from EIN-ring membership of support vector pairs. Moreover, the outliers in the training data are automatically eliminated according to their EIN-ring membership in the step of finding region components. The candidate support vectors are selected in a way that is robust to noisy and fuzzy boundary. Experiments show that P-SVM is order of magnitude faster than traditional SVM with superior cardinality scalability and comparable accuracy for a large-scale spatial data.

Keywords: Support Vector Machine, Data mining, Peano trees, EIN-rings.

1 INTRODUCTION

Data mining for spatial data, e.g., remote sensing, geographical information systems, astronomy, computer cartography, environmental assessment and planning, etc., is a rapidly evolving research area in recent years. Support vector machine (SVM) as a powerful tool for data classification has potential application for spatial data mining.

The theory of SVM was first introduced by Vapnik back in the sixties [1]. It solves the discrimination hyperplane by a mathematical programming problem with an objective function, which balances between maximal separation and errors in linear inseparable case. SVM has empirically been shown to give good generalization performance on a wide variety of problems [2][3][4][5][6]. Although SVM has good generalization performance, it involves solving a quadratic program that require considerably long computation time, which makes it difficult for large spatial data.

In this paper, we propose a new efficient proximal Support Vector Machine (P-SVM) using Peano tree. Peano tree is a lossless vertical compressed bitwise quadrant tree that is processed horizontally through logical operations [7][8]. This approach stands in contrast to the ubiquitous approach of vertically scanning horizontal data structures (records). P-SVM exploits a unique neighborhood search method, i.e., EIN-ring based neighborhood search to find the boundary sentries. The final boundary hyperplanes of test data are determined by their \(d\)-nearest boundary sentries, which are calculated from EIN-ring membership of support vector pairs. Moreover, the outliers in the training data are automatically eliminated according to their EIN-ring membership in the step of finding region components. The candidate support vectors are selected in a way that is robust to noisy and fuzzy boundary. The overall average run time complexity for spatial data is \(O(d^*n \sqrt{n})\), where \(d\) is the dimension and \(n\) is the size of data set. Experiments show that P-SVM is order of magnitude faster than traditional SVM with superior cardinality scalability and comparable accuracy for a large-scale spatial data.

This paper is organized as follows. In section 2, we review the basic P-tree and its variation, range predicate tree. In section 3, we describe a new efficient proximal support vector machine using P-trees. Finally, we compare our method with traditional \(C\)-SVM methods experimentally in section 4 and conclude the paper in section 5.

2 REVIEW OF PEANO TREES

A new tree structure, the Peano tree (P-tree), was developed to facilitate efficient data mining [7]. In this section, we first briefly review the basic P-trees, and then describe a new calculation method of a variation of P-tree, range predicate trees. In this paper, we use \(\wedge, \vee\) and prime (‘) to denote P-tree operations AND, OR and NOT, respectively.
2.1 Basic Peano Trees

A basic P-tree is a lossless, bitwise, vertical quadrant-based compressed tree, which can be 1-dimensional, 2-dimensional, 3-dimensional, etc. The recursive raster ordering is called the Peano or Z-ordering in the literature – therefore, the name Peano tree. One quadrant of a two-dimensional P-tree with four sub-quadrants, quadrant 0, 1, 2, and 3, is shown in Figure 1.

![Figure 1 Peano Ordering or Z Ordering](image)

For a data set with d feature attributes, \( X = (A_1, A_2 \ldots A_d) \), and the binary representation of \( j^{th} \) feature attribute \( A_j \) as \( b_{j,m}b_{j,m-1} \ldots b_{j,i}b_{j,0} \), we strip each feature attribute into several files, one file for each bit position. Such files are called bit files. A bit file is recursively partitioned into quadrants and each quadrant into sub-quadrants until the sub-quadrant is pure (entirely 1-bits or entirely 0-bits).

We illustrate the detailed construction of P-trees using an example shown in Figure 2. The spatial data is the red reflective value of an 8x8 2-dimensional spatial data, which is shown in a). We represent the reflectance as binary values, e.g., \((7)_{10} = (111)_{2}\). Then strip them into three separate bit files, one file for each bit, as shown in b), c), and d). The corresponding basic P-trees, \( P_1, P_2 \) and \( P_3 \), are constructed by recursive partition, which are shown in e), f) and g).

As shown in e) of Figure 2, the root of \( P_1 \) tree is 36, which is the 1-bit count of the entire bit file-1. The second level of \( P_1 \) contains the 1-bit counts of the four quadrants, 16, 7, 13, and 0. Since quadrant 0 and quadrant 3 are pure, there is no need to partition these quadrants. Quadrant 1 and 2 are further partitioned recursively.

AND, OR and NOT logic operations are the most frequently used P-tree operations. The P-tree logical operations are performed level-by-level starting from the root level. They are commutative and distributive, since they are simply pruned bit-by-bit operations. For instance, ANDing a pure-0 node with anything results in a pure-0 node, ORing a pure-1 node with anything results in a pure-1 node.

2.2 Range Predicate Trees

Range predicate tree, \( P_{x < y} \), is a basic P-tree that satisfies predicate \( x < y \), where \( y \) is a boundary value, and \(< \) is the comparison operator, i.e., \(<, >, \geq, \) and \( \leq \). Without loss of generality, we only describe the calculation of range predicate \( P_{A < c} \) and \( P_{A > c} \) as follows.

\[
P_{A < c} = P_m \text{op}_m \ldots P_i \text{op}_i \ldots \text{op}_{k+1} P_k, \quad k \leq i \leq m, \quad (1)
\]

where 1) \( \text{op}_i \) is \( \wedge \) if \( b_i=1 \), \( \text{op}_i \) is \( \vee \) otherwise, 2) \( k \) is the rightmost bit position with value of “0”, i.e., \( b_k=0 \), \( b_{k-1}=1, \forall j < k \), and 3) the operators are right binding. Here the right binding means operators are associated from right to left, e.g., \( P_2 \text{op}_2 P_1 \text{op}_1 P_0 \) is equivalent to \( (P_2 \text{op}_2 (P_1 \text{op}_1 P_0)) \).

Let \( P_{A \leq c} \) be the range predicate tree for \( A \leq c \), then

\[
P_{A \leq c} = P_m \text{op}_m \ldots P_i \text{op}_i \ldots \text{op}_{k+1} P_k, \quad k \leq i \leq m, \quad (2)
\]
where 1. \( \text{op}_i \) is \( \wedge \) if \( b_i=0 \), \( \text{op}_i \) is \( \lor \) otherwise, 2) \( k \) is the rightmost bit position with value of “0”, i.e., \( b_k=0 \), \( b_j=1 \), \( \forall j<k \), and 3) the operators are right binding. For a data set with \( d \) feature attributes, \( X = (A_1, A_2 \ldots A_d) \), the tuple range predicate tree is calculated as

\[
P_X \prec \gamma = P_{A1} \prec c_1 \land \ldots \land P_{A_j} \prec c_j \ldots P_{A_d} \prec c_d.
\]

3 PROXIMAL SUPPORT VECTOR MACHINE

In this chapter, we propose a Proximal Support Vector Machine (P-SVM) using P-tree. Our main idea is to fit the classification boundary using segment hyperplanes in original data space. The P-SVM includes four steps. 1) Locating region components by partitioning the training data into region components using equal interval neighborhood ring (EIN-ring). 2) Finding support vectors by calculating EIN-ring membership of a data set in region component. 3) Fitting the boundary by calculating \( d \)-nearest boundary sentries of test data. 4) The class label of the test data is determined by its location relative to the boundary hyperplane. The details of each step are described in the following subsections.

3.1 Finding Region Components

In this section, we first define the EIN-ring and region components, and then give the algorithm of finding region components using EIN-ring.

**Definition 1. The Neighborhood Ring** of data point \( c \) with radii \( r_1 \) and \( r_2 \) is defined as the set \( R(c, r_1, r_2) = \{ x \in X \mid r_1 \leq |c-x| \leq r_2 \} \), where \( |c-x| \) is the distance between \( x \) and \( c \).

**Definition 2. The Equal Interval Neighborhood Ring** of data point \( c \) with radii \( r \) and fixed interval \( \lambda \) is defined as the neighborhood ring \( R(c, r, r+\lambda) = \{ x \in X \mid r \leq |c-x| \leq r+\lambda \} \), where \( |c-x| \) is the distance between \( x \) and \( c \).

The interval \( \lambda \) is a user-defined parameter based on accuracy requirements. The higher the accuracy requirement, the smaller the interval. For \( r = k\lambda \), \( k=1,2,\ldots \), the rings called the \( k^{th} \) EIN-rings. Figure 3 shows 2-D EIN-rings with \( k = 1, 2 \), and 3.

**Definition 3. Region Component** Given a training data set \( X \) with \( C \) classes, region components are groups of the training data points, \( x \), which have more than half classmates within \( \sigma \) neighbors, where classmates are the data points with same class label as \( x \) and \( \sigma \) is a numeric threshold of neighbors.

**Figure 3** Diagram of EIN-rings

Finding neighbors within a EIN-ring ring, which is accomplished by range predicate tree, is the first step to find the region components. The calculation of neighbors within EIN-ring \( R(x, r, r+\lambda) \) is as follows. Let \( P_{r,\lambda} \) be the P-tree representing data points within EIN-ring \( R(x, r, r+\lambda) \). We first set \( P_{r,\lambda} \) as the predicate tree corresponding to the predicate \( x-r-\lambda \leq X \leq x+r+\lambda \). We first calculate the data points within neighborhood ring \( R(x, 0, r) \) and \( R(x, 0, r+\lambda) \) by \( P_{x-r<X} \) and \( P_{x-r-X<r+X<} \) respectively. The data points within the EIN-ring \( R(x, r, r+\lambda) \) are those that are in \( R(x, 0, r+\lambda) \) but not in \( R(x, 0, r) \). Therefore \( P_{r,\lambda} \) is calculated by the following formula

\[
P_{r,\lambda} = P_{x-r<X} \land P_{x-r-X<r+X<}
\]

Let \( P_{r,c} \) be the class label P-tree, which is built for class \( i \) within the data set \( X \). A “1” value in \( P_{r,c} \) indicates that the corresponding data point has class label \( i \). A “0” value in \( P_{r,c} \) indicates that the corresponding data point does not have class label \( i \). The number of neighbors with the EIN-ring, \( R(x, r, r+\lambda) \), is

\[
\text{NBR}_x = \text{RootCount} (P_{r,\lambda} \land P_{r,c}).
\]

We set NBRx as a fixed number \( \sigma \), e.g., 4, 6, or 8. If \( \text{NBR}_x < \sigma \), decrement \( r \) until \( \text{NBR}_x \geq \sigma \). Check the neighbors of \( x \), mark \( x \) the same group as its classmates within the neighborhood. If none of its classmates within the neighborhood are marked, mark \( x \) as a new group member. If the number of \( x \)’s classmates is less than \( \sigma/2 \), treat \( x \) as an outlier and zero it. After all data objects are proceeded and labeled with group ID, merge groups which have the same class label and are reachable to each other within \( \sigma \) neighbors. Finally, code the remaining group ID as a simple binary number and build corresponding group ID P-trees, \( P_{gi} \). The \( P_{gi} \) will be used to find support vector pairs which will be discussed in the later section. The algorithm for finding region components is given in Figure 4.
3.2 Finding Support Vector Pairs

We first define EIN-ring membership and EIN-ring membership pair of data x in the region component g. They are used to find support vector pairs around the boundary. Then we give the algorithm of finding support vector pairs using tuple-P-trees ANDing.

**Definition 3.2.1 EIN-ring Membership** The EIN-ring membership of data x in the region component g, $M_{xg}$, is defined as normalized summation of the weighted tuple-P-tree root counts within EIN-ring, $R(x, r, r+\lambda)$, which is calculated as follows

$$M_{xg} = \frac{1}{N_g} \sum_{r=1}^{s} w_r * NBR_{xg,r}$$  \hspace{1cm} (6)

where $N_g$ is the number of data points in region component g, $NBR_{xg,r}$ is the number of neighbors in group g, and $w_r$ is the weight of the EIN-ring, $R(x, r, r+\lambda)$.

The number of neighbors in group g within the EIN-ring, $NBR_{xg,r}$, is calculated as

$$NBR_{xg,r} = \text{RootCount} (P_{xg})$$  \hspace{1cm} (7)

where $P_{xg}$ is the P-tree of data point x in region component g within the EIN-ring, $R(x, r, r+\lambda)$, which is calculated as

$$P_{xg} = P_{i\lambda} \land P_{g}$$  \hspace{1cm} (8)

Where $P_g$ is group ID P-tree. There are many weighting functions that can be used to adjust the EIN-ring membership by weights. The selection of this weight is based on a RBF kernel function of the radius of EIN-ring, such as Gaussian function, step function, etc.

**Definition 3.2.2. EIN-ring Membership Pair**

The EIN-ring membership pair of data x in the region component g, $\text{HMP}_{xg}$, is defined as $\text{HMP}_{xg} = (M_{xg}, M_{xg'})$, where g' is the neighboring region component of data x.

EIN-ring Membership Pair of data x, $\text{HMP}_{xg}$, indicates the location of x relative to the boundary of region component g and g'. According to the definition of $\text{HMP}_{xg}$, $M_{xg} > M_{xg'}$ for x which is far away from the boundary. If x is around the boundary, then $M_{xg} = M_{xg'}$.

We define that data point x, x belonging to group g, is a candidate data support vector if $M_{xg} > M_{xg'}$. The candidate data support vectors are those that are at the right side of boundary.

**Definition 3.2.3. Support Vector Pair** A pair of candidate data support vectors, $x_i, x_j \in X, i \neq j$, is the support vector pair, $\text{SVP}(x_i, x_j)$, if and only if $d(x_i, x_j) \leq d(x_i, x_k) \forall x_k \in X$ and $x_k \in g, x_i \in g'$. In other words, $x_i$ and $x_j$ are at the right side of the boundary and the nearest neighbors from the different region components. The pseudo code of finding support vector pairs is given in Figure 5.
for the boundary hyperplane of the test data, which is defined as follows.

**Definition 3.3.1. Boundary Sentry** The boundary sentry, BS\(_{ij}\), of a support vector pair, SVP\((x_i, x_j)\), is defined as

\[
BS_{ij} = \lambda x_i + (1 - \lambda)x_j,
\]

where \(\lambda = M_{s<e} / (M_{s<e} + M_{s>e})\). \(M_{s<e}\) is the EIN-ring membership of data \(x_i\) in group \(g\), and \(M_{s>e}\) is the EIN-ring membership of data \(x_j\) in group \(g'\).

**Algorithm 3.3.1. Fitting Boundary Hyperplane**

Given a test data point \(x\), the boundary hyperplane is specified by a vector \(w \in \mathbb{R}^d\) and a bias \(w_0\) as

\[
H(x) = wx + w_0,
\]

where the vector, \(w\), and bias, \(w_0\), is solved by the \(d\)-nearest boundary sentries. For example, if \(d = 2\), the boundary is a line and we need two boundary sentries, BS\(_{ij}\) to determine it, where \(x_i \in g\) and \(x_j \in g'\) for all corresponding SVP \((x_i, x_j)\). Similarly, if \(d = 3\), we need three boundary sentries to determine the boundary plane, where \(x_i \in g\) and \(x_j \in g'\) for all corresponding SVP \((x_i, x_j)\).

After fitting the boundary hyperplane for the test data \(x\), we need to determine the class label of \(x\), which is determined by relative location to the boundary hyperplane. The location of data point \(x\) relative to boundary hyperplane is calculated as

\[
\text{Sign}(x) = \text{Sign}(H(x) - wx + w_0),
\]

where \(\text{Sign}\) is a function to get the sign of \(x\). If \(\text{Sign}(x) = \text{Sign}(x_i)\), where \(x_i \in g\), then \(x \in g\), otherwise \(x \in g'\). Then we will find the class label of \(x\) according to group. The algorithm for finding the boundary hyperplane of test data, \(x\), is given in Figure 6.

Here is a brief complexity analysis of our algorithm. Let \(f\) be the fan-out of a P-tree, \(n\) be the size of the data set, and \(d\) be the dimension. The total number of nodes in a P-tree with a compression ratio of \(\rho < 1\) is \(\eta = 1 + (\rho^k \ast n - f) / (f \ast \rho - 1)\), where \(k\) is the number of levels of P-tree.

The total run time at the worst case to find the nearest neighbors of \(n\) data points in \(d\)-dimension is \(d^m n \ast \eta\), where \(\eta\) is the total number of nodes of a P-tree, \(\eta = (n - 1) / (f - 1)\). Thus, the run time complexity is \(O(dn^2)\) at the worst case.

For data sets with fan-out \(f = 4\) and average compression rate \(\rho = 0.5\), the total number of nodes of a P-tree is \(\eta = 1 + (\sqrt{n} - 8) / 2\). Therefore, the total time at the average case to find the nearest neighbors of \(n\) data points in \(d\)-dimension is \(d^m n \ast \eta = (1 + (\sqrt{n} - 8) / 2)\), which is \(O(dn)\).

**Figure 6** Algorithm of Finding Boundary Sentry

The P-SVM algorithm includes four steps as described above. The main cost of step one and step two is at finding the nearest neighbors of \(n\) data points. The average run time complexity of finding the nearest neighbors of \(n\) data points with compression ratio 0.5 and fan-out 4 of P-tree is \(O(d^m n \ast \sqrt{n})\). The run time complexity is \(O(dn^2)\) at the worst case. The run time complexity of step three is proportional to the number of SVPs. Since the number of SVPs is far less than \(n\), the run time complexity of step three can be ignored. Thus, the overall average run time complexity of the P-SVM algorithm for spatial data is \(O(dn \ast \sqrt{n})\).

**4 PERFORMANCE ANALYSIS**

Our experiments were implemented in the C++ language on a 1GHz Pentium PC machine with 1GB main memory, and Debian Linux 4.0. The test data includes the aerial TIFF image with red, green and blue attribute reflectance values, moisture map, nitrate map, and yield.
map of the Oakes Irrigation Test Area in North Dakota. The TIFF image size is 1320x1320. The data sets are available at [9]. Among these six attributes we consider yield as the class attribute. Each attribute is 8 bits long. So we have 8 basic P-trees for each attribute and 40 (for the 5 attributes excluding yield) in total. For the class attribute, yield, we considered only the most significant 3 bits. Therefore we have eight different class labels for the pixels. We built 8 value P-trees from the yield values – one for each class label. In each experiment run, we randomly select the 10% of data set as test data and the rest as training data. Table 1 shows the experiment results of average error rate for 30 runs of P-SVM and C-SVM implemented by SVMlight [10].

Table 1.  Performance Comparison

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Testing Correctness %</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = # of pixels</td>
<td>P-SVM</td>
</tr>
<tr>
<td>n = 16x16</td>
<td>86.4%</td>
</tr>
<tr>
<td>n = 32x32</td>
<td>89.0%</td>
</tr>
<tr>
<td>n = 64x64</td>
<td>90.3%</td>
</tr>
<tr>
<td>n = 128x128</td>
<td>92.0%</td>
</tr>
<tr>
<td>n = 256x256</td>
<td>94.1%</td>
</tr>
<tr>
<td>n = 512x512</td>
<td>86.4%</td>
</tr>
</tbody>
</table>

As shown in Table 1, the testing correctness of P-SVM and C-SVM on all the five dataset is almost identical. It indicates that the P-SVM has comparable accuracy with C-SVM. In order to compare cardinality scalability, we formed 16x16, 32x32, 64x64, 128x128, 256x256, and 512x512 images by choosing pixels that are uniformly distributed in original image. The average CPU run time of 30 runs on the five different sizes of data is shown in Figure 7.

![Figure 7 Running Time Comparison](image)

It is shown that the P-SVM is faster than C-SVM on all five different sizes of data set. When the data set size increases, the run time of P-SVM method increases at a much lower rate than C-SVM. The experiment results show that P-SVM method is more scalable for large spatial data set.

5 CONCLUSION

In this paper, we propose a new efficient proximal Support Vector Machine (P-SVM) using Peano tree. This approach stands in contrast to the ubiquitous approach of vertically scanning horizontal data structures (records). The overall average run time complexity for spatial data is $O(d^n \sqrt{n})$, where d is the dimension and n is the size of data set. Experiments show that P-SVM is order of magnitude faster than traditional SVM with superior cardinality scalability and comparable accuracy for a large-scale spatial data.

REFERENCES