A Parallel Genetic Algorithm for Solving the Inverse Problem of Support Vector Machines

Qiang He, Xizhao Wang, Junfen Chen, and Leifan Yan

Faculty of Mathematics and Computer Science,
Hebei University, Baoding 071002, Hebei, China
{heq, wangxz}@mail.hbu.edu.cn

Abstract. Support Vector Machines (SVMs) are learning machines that can perform binary classification (pattern recognition) and real valued function approximation (regression estimation) tasks. An inverse problem of SVMs is how to split a given dataset into two clusters such that the maximum margin between the two clusters is attained. Here the margin is defined according to the separating hyper-plane generated by support vectors. This paper investigates the inverse problem of SVMs by designing a parallel genetic algorithm. Experiments show that this algorithm can greatly decrease time complexity by the use of parallel processing. This study on the inverse problem of SVMs is motivated by designing a heuristic algorithm for generating decision trees with high generalization capability.

1 Introduction

Support vector machines (SVMs) are a classification technique of machine learning based on statistical learning theory [1, 2]. Considering a classification problem with two classes, SVMs are used to construct an optimal hyper-plane that maximizes the margin between two classes. According to Vapnik statistical learning theory [1, 3], the maximum of margin implies an extraordinary generalization capability and good performances of SVM classifiers [4, 5]. So far, SVMs have already been successfully applied to many real fields. This paper aims to make preparation for SVM's application to decision tree generation.

Given a training set, a general procedure for generating a decision tree can be briefly described as follows:

The entire training set is first considered as the root node of the tree. Then the root node is split into two sub-nodes based on appropriate heuristic information. If the instances in a sub-node belong to one class, then the sub-node is regarded as a leaf node, else we continue to split the sub-node based on the heuristic information. This process repeats itself until all leaf nodes are generated. The most popular heuristic information used in the decision tree generation is the minimum entropy. This heuristic information has many advantages such as small leaf numbers and less computational effort. However, it has a serious disadvantage – the poor generalization capability.

D.S. Yeung et al. (Eds.): ICMLC 2005, LNAI 3930, pp. 871 – 879, 2006.

The investigation into the inverse problem of SVMs is motivated by designing a new decision tree generation procedure to improve the generalization capability of existing decision tree programs based on minimum entropy heuristic. Due to the relationship between the margin of SVMs and the generalization capability, the split with maximum margin may be considered as the new heuristic information for generating decision trees.

This paper has the following organization; Section 2 briefly reviews the basic concept of support vector machines. Section 3 proposes the inverse problem of SVMs and designs a parallel genetic algorithm to solve this problem. Section 4 gives some experiment results to demonstrate the feasibility and effectiveness of the parallel genetic algorithm, especially in the way of time complexity. And the last section briefly concludes this paper.

2 Support Vector Machines

2.1 The Basic Problem of SVMs

Let $S = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$ be a training set, where $x_i \in R^n$ and $y_i \in \{-1, 1\}$ for $i = 1, 2, \cdots, N$. The optimal hyper-plane of S is defined as $f(x) = 0$, where

$$f(x) = (w_0 \cdot x) + b_0 \tag{1}$$

$$w_0 = \sum_{j=1}^{N} y_j \alpha_j^0 x_j \tag{2}$$

$(w_0 \cdot x) = \sum_{i=1}^{n} w_0^i \cdot x^i$ is the inner product of the two vectors, where $w_0 = (w_0^1, w_0^2, \cdots, w_0^n)$ and $x = (x^1, x^2, \cdots, x^n)$. The vector w_0 can be determined according to the following quadratic programming [1]

$$\begin{cases} \text{Maximum } W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \\ \text{Subject to } \sum_{j=1}^{N} y_j \alpha_j = 0; C \geq \alpha_i \geq 0, \ i = 1, 2, \cdots, N \end{cases} \tag{3}$$

where C is a positive constant. The constant b_0 is given by

$$b_0 = y_i - \left(x_i \cdot \sum_{j=1}^{N} y_j \alpha_j^0 x_j\right) \tag{4}$$

Substituting (2) for w_0 in (1), we have

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i^0 (x_i \cdot x) + b_0 \tag{5}$$

We can identify separability of two subsets by checking whether the following inequalities

$$y_i\left(w_0 \cdot x_i + b\right) \geq 1;\ \ i = 1, 2, \cdots, N \tag{6}$$

hold well[1].

A procedure to compute a maximum margin for two subsets is described below.

Procedure 1. The constant C in equation (3) is selected to be large at first.

Step 1. Solve the quadratic programming (3).
Step 2. Determine the separating hyper-plane (5) according to (4).
Step 3. Check the separability between two subsets according to inequalities (6).
Step 4. Let the margin be 0 if the two subsets are not separable.
Step 5. Compute the maximum margin according to $1/\left(w_0 \cdot w_0\right)$ for the separable case where the vector w is determined by (2).

2.2 Generalization in Feature Space

In practice, the performance of SVMs based on the previous section may not be very suitable for the nonlinear-separable cases in the original space. To improve the performance and to reduce the computational load for the nonlinear separable datasets, Vapnik [1] extended the SVMs from the original space to the feature space. The key concept of the extension is that a SVM first maps the original input space into a high-dimensional feature space through some nonlinear mapping, and then constructs an optimal separating hyper-plane in the feature space. Without any knowledge of the mapping, the SVM can find the optimal hyper-plane by using the dot product function in the feature space. The dot function is usually called a kernel function. According to the Hilbert-Schmidt theorem [1], there exists a relationship between the original space and its feature space for the dot product of two points. That is

$$(z_1 \cdot z_2) = K(x_1, x_2) \tag{7}$$

where it is assumed that a mapping Φ from the original space to the feature space exists, such that $\Phi(x_1) = z_1$ and $\Phi(x_2) = z_2$, and $K(x_1, x_2)$ is conventionally called a kernel function satisfying the Mercer theorem [1]. Usually the following three types of kernel functions can be used: polynomial with degree p, radial basis function and sigmoid function [1]. Replacing the inner product $\left(x_1 \cdot x_2\right)$ in (5) with the kernel function $K\left(x_1, x_2\right)$, the optimal separating hyper-plane becomes the following form:

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i^0 K\left(x_i, x\right) + b_0 \tag{8}$$

It is worth noting that the conclusion of section 2.1 is still valid in the feature space if we substitute $K\left(x_1, x_2\right)$ for the inner product $\left(x_1 \cdot x_2\right)$.

3 An Inverse Problem of SVMs and Its Solution Based on Genetic Algorithms

For a given dataset for which no class labels are assigned to instances, we can randomly split the dataset into two subsets. Suppose that one is the positive instance subset and the other is the negative instance subset, we can calculate the maximum margin between the two subsets according to Procedure 1 where the margin is equal to 0 for the non-separable case. Obviously, the calculated margin depends on the random split of the dataset. Our problem is how to split the dataset such that the margin calculated according to Procedure 1 attains the maximum.

It is an optimization problem. We mathematically formulate it as follows:

Let $S=\{x_1, x_2, \cdots, x_N\}$ be a dataset and $x_i \in R^n$ for $i=1,2,\cdots,N$, $\Omega=\{f \mid f \text{ is a function from } S \text{ to } \{1,-1\}\}$. Given a function $f \in \Omega$, the dataset can be split into two subsets and the margin can then be calculated by Procedure 1. We denote the calculated margin (the functional) by $\text{Margin}(f)$. Then the inverse problem is formulated as

$$\text{Maximum}_{f\in\Omega}\left(\text{Margin}(f)\right) \tag{9}$$

Due to the exponentially increased complexity, it is not feasible to enumerate all possible functions in Ω for calculating their margins according to Procedure 1. It is difficult to give an exact algorithm for solving the optimization problem (9). Here we can design a genetic algorithm to solve (9).

First, we briefly review the main steps of a general simple genetic algorithm [9].

Procedure 2. A general procedure of genetic algorithms for solving an optimization problem with several variables:

Step 1. Determine the encoding mechanism for representing the optimization problem's variables.

Step 2. Initialize the population, which contains a number of encoded samples (called chromosomes) based on the encoding mechanism.

Step 3. Specify the fitness function, which normally takes the values in [0, 1] and is defined in the set of all chromosomes.

Step 4. Select parents (chromosomes) from the current population according to their fitness values.

Step 5. Produce their offspring via the crossover operation, which usually means to partially exchange genes of two parent chromosomes.

Step 6. Conduct mutation operation, i.e., genes of the offspring chromosome change with a certain probability.

Step 7. Consider all offspring as the new population and check whether a termination criterion is reached. If yes, go to step 8, else, go to step 4.

Step 8. Stop.

Then, we can design a genetic algorithm to solve the proposed inverse problem of SVMs according to the above Procedure 2.

Procedure 3. A general procedure of genetic algorithms for solving the proposed inverse problem of SVMs:

Step 1. Each function $f \in \Omega$ corresponds to a binary partition of the dataset S. Therefore each f can be viewed as a N-dimensional vector such as $100011101\cdots01$ with N bits. Each bit taking value 0 or 1 is regarded as a gene corresponding to an instance in S. Thus, each chromosome (a bit string such as $100011101\cdots01$) consisting of N genes represents a function in Ω where, if a bit is 1, it means that the corresponding instance is positive; and a value 0 represents that the corresponding instance is negative. The fixed length of each chromosome's coding is N, the number of instances of the initial dataset.

Step 2. Given an integer M denoting the size of the population, uniformly generate N random numbers (0 or 1), which constitute a chromosome. Repeat M times and hence generate M chromosomes.

Step 3. Noting that each chromosome can determine a training set given in Section 2, we define the fitness value for each chromosome as the margin value computed by Procedure 1. Here the fitness value is 0 if the chromosome corresponds to a non-separable training set, and is the real margin of the SVM if the chromosome corresponds to a separable training set.

Step 4. Reproduction. This is a process in which individual strings are copied in terms of their fitness values. In traditional textbook manner, the reproduction is conducted by a technique of roulette-wheel parent selection, which indicates that the probability with which an individual is selected is proportional to its fitness value. This technique can be implemented algorithmically as follows [7]:

(1) Let the population be $\{1,2,\cdots,M\}$ and $f(j)$ denotes the fitness value of the j-th individual. Compute $s_i = \sum_{j=1}^{i} f(j)\ (i = 1,2,\cdots,M)$.

(2) Generate a random number α uniformly distributed in the interval [0, s_M].

(3) Return the first individual whose fitness value plus the values of fitness of the previous individuals are greater than or equal to α. That is, this step returns the k-th individual with the property $s_{k-1} < \alpha \le s_k$.

The reproduction is used to generate M parent candidates. We suppose that the M parent candidates contain the individual with the highest fitness. (If not, we can specify the individual with the highest fitness as a candidate).

Step 5. Crossover. Reproduction results in a mating pool consisting of M parent candidates. Then $M/2$ pairs of parents are randomly selected from the pool. The crossover site (a bit position) is also selected randomly. The crossover happens with probability p_c for each selected pair. This crossover operator leads to $M/2$ pairs of offspring, i.e., M new chromosomes.

Step 6. Mutation. It means that a bit of an offspring chromosome is replaced with a randomly chosen bit. The mutation is performed with probability p_m for each chromosome.

Step 7. Calculate the M parents' fitness values and place them with their M children to form a set of 2M chromosomes. Sort the 2M chromosomes based on their fitness values from big to small, and then choose the first M chromosomes with the highest fitness values as the population of the next generation.

Step 8. The predefined maximum number of generations, T, is chosen as the termination criterion. If the generation number is less T then go to Step 4; else go to Step 9.

Step 9. Output the first chromosomes and its fitness value. According to this chromosome, in which value 1 corresponds to a positive instance and value 0 corresponds to a negative instance, the final partition (split) of the initial dataset is obtained. And the outputted fitness value is the maximum margin. Stop.

The decision function f obtained through the above algorithm denotes the optimal or approximately optimal solution of problem (9) when the parameters in GA are selected properly. In addition, it is worth mentioning that genetic algorithms cannot be guaranteed to obtain the optimal solution every time, so it is expected they will have a big probability for obtaining the optimal or approximately optimal solution. To raise the probability of obtaining the optimal solution, one needs to increase the population size or the maximum number of generations, which obviously is at the price of increased running time.

One main reason that the proposed genetic algorithm has large time complexity is the process of solving quadratic programs; that is, calculating each chromosome's fitness value. How to reduce the time complexity of the algorithm (for large databases especially) is a very important issue to be investigated. Here we use the method of parallel processing on Linux Clusters to solve this problem.

Procedure 4. A parallel procedure of genetic algorithms for solving the proposed inverse problem of SVMs:

Step 1. Choose the same encoding mechanism as Procedure 3. Specify the penalty factor C, the maximum number of generations MaxG, the population size M, the crossover probability p_c, the mutation probability p_m and the fixed length of each chromosome's coding N, which is the number of instances of the initial dataset.

Step 2. Let the master process generate M chromosomes as the first population on the master node of Linux Clusters.

Step 3. Calculate each chromosome's fitness value on slave nodes by parallel method, that is, the margin value computed by Procedure 1. We describe it in detail by pseudo codes as follows:

for i = 1 to *numtask* par-do
    calculate each chromosome's fitness value;
end for

This is a parallel statement, where *numtask* is the number of chromosomes processed by every slave process on the corresponding slave node, and i is the number of slave processes (slave node). It is important that a synchronization mechanism is used in order to avoid any problems when parallel computation is completed.

Step 4. Reproduction. This is the same as Procedure 3; however, it is only done on the master node. The reproduction will generate another set of M parent candidates, which are put into the buffer called a mating pool.

Step 5. Crossover. This operation is just for the M candidates from the mating pool, and is done on the master node.

Step 6. Mutation. It is also for the M candidates and is done on the master node.

Step 7. Calculate the M candidates' fitness values as in Step 3. In succession, sort the 2M chromosomes based on their fitness values from big to small, and then choose the first M chromosomes with the highest fitness values as the population of the next generation.

Step 8. If the generation number is less MaxG then go to Step 4; else go to Step 9.

Step 9. Output the final result, the first chromosome and its fitness value, i.e., the best chromosome and the maximum margin. Stop.

Here our parallel algorithm is a global single-population master-slave genetic algorithm [8]. In a master-slave genetic algorithm there is a single population (just as in a simple genetic algorithm), but the process to get each chromosome's fitness value, which consumes more time, is distributed among slave nodes and done by means of parallel processing. Since in this type of parallel genetic algorithm, selection and crossover consider the entire population, it is also known as a global parallel genetic algorithm.

4 Experimental Results

Experimental environment refers to Table 1(a) & (b).

To verify the effectiveness of the parallel genetic algorithm, we construct a small dataset with 20 2-dimensional points (Table 2). The parameters specified in the parallel algorithm are shown in Table 3. Table 4 is the experimental results of the above dataset on the original space for the parallel genetic algorithm, which shows the relationship between the running time and the number of computing nodes. From Table 4 one can see that the running time of the parallel genetic algorithm is significantly reduced with the number of computing nodes increased.

A well-known dataset called Iris [9] is selected to verify the advantage of the parallel algorithm. We used 50 samples of the dataset (25 from the second class and another 25 from the third class) for the verification. Table 5 shows the running time change with the increase of computing nodes. From Table 5 we observe that the running time rapidly decreases with the computing nodes. The decrease is significant, because the process to get each chromosome's fitness value, which has larger time complexity, is done by means of parallel processing.

Table 1(a). Node devices configuration

CPU	Intel Pentium 4 Xeon 3.06GHz ×2
Memory	512MB DDR
Bus	PCI-X
Disk	80G IDE

Table 1(b). Cluster configuration

No. of computing nodes	16
No. of management nodes	2
Network	100M-Ethernet, 2G-Myrinet
Operating system	Redhat 9.0
Programming environment	MPICH

Table 2. A small dataset

Case	Feature1	Feature2	Case	Feature1	Feature2
1	0.116	0.710	11	0.422	0.306
2	0.248	0.860	12	0.574	0.396
3	0.362	0.798	13	0.748	0.308
4	0.254	0.642	14	0.560	0.194
5	0.116	0.532	15	0.598	0.308
6	0.150	0.852	16	0.656	0.512
7	0.188	0.760	17	0.626	0.562
8	0.282	0.750	18	0.766	0.436
9	0.168	0.640	19	0.780	0.562
10	0.358	0.640	20	0.666	0.398

Table 3. Parameters in genetic algorithm

POPSIZE=90	Size of population
PC=0.7	Probability of crossover
PM=0.6	Probability of mutation
NB=0.3	Gen mutation proportion
MAXGENERATION=20	Maximum generation
C=100	Penalty factor

Table 4. Experimental results on the original space of the dataset (Table 2)

No. of computing nodes	Time (minutes)	The best margin
3	9.567	0.421
6	5.825	0.421
15	4.507	0.421

Table 5. Running time with an increase of computing nodes in the Iris dataset

No. of computing nodes	3	6	9	10	15
Time (minutes)	69.971	35.185	28.956	27.125	22.048

5 Concluding Remarks

Motivated by design of a new heuristic procedure of generating decision trees with higher generalization capability, a genetic algorithm can be used to solve an inverse problem of SVMs, but its time complexity is larger. To overcome this disadvantage, this paper proposes an improved version, the parallel genetic algorithm, which can reduce time complexity significantly.

Acknowledgement

This paper is supported by the National Natural Science Foundation of China(Project No. 60473045) and the Young Research Fund of Hebei University.

References

1. V. N. Vapnik, Statistical learning theory, New York, Wiley, 1998, ISBN: 0-471-03003-1
2. V. N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 2000, ISBN: 0-387-98780-0
3. V. N. Vapnik, An Overview of Statistical Learning Theory, IEEE Transactions on Neural Networks, Vol. 10, No.5, Pages 88 - 999, 1999.
4. B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in Proc. Fifth Annual Workshop on Computational Learning Theory. ACM Press, Pittsburgh. Guyon, D. Haussler, Ed., 1992, pp. 144-152.
5. R. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," Ann. Statist., vol. 26, no. 5, pp. 1651–1686, 1998
6. J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony, "Structural risk minimization over data-dependent hierarchies," IEEE Trans. Inform. Theory, vol. 44, pp. 1926–1940, Sept. 1998
7. Chin-Teng Lin and C.S. George Lee, Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems, Prentice Hall PTR, 1996, 797 pages ,ISBN 0-13-235169-2.
8. Erick Cantú-Paz, "A survey of parallel genetic algorithms," Tech. Rep., The University of Illinois, 1997, IlliGAL Report No. 97003, FTP address: ftp://ftpilligal.ge.uiuc.edu/pub/papers/IlliGALs/97003.ps.Z.
9. UCI Repository of machine learning databases and domain theories. FTP address: ftp://ftp.ics.uci.edu/pub/machine-learning-databases.