2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.2.47

An FPGA Implementation of Multi-Class Support Vector Machine Classifier Based on Posterior Probability

Zhiliang Nie+, Xingming Zhang and Zhenxi Yang

National Digital Switch System Engineering & Technological R & D Center, 450002

Zhengzhou, Henan, China

Abstract—Aim at the issue that the results of analyzing a classify algorithm of multi-class support vector machines which is based on posterior probability have been found its computation complexity is large, this paper present a simplified algorithm which is suitable for FPGA implementation. The analysis results indicate that this simplified algorithm's computation complexity is decline. The experimental results in language recognition system show that the mean absolute error between the fixed-point output of the FPGA-based simplified algorithm and the floating point output of the original algorithm which is based on C language is 10-4 order of magnitude, which hardly affect this system's performance and meet its real-time requirements.

Keywords: posterior probability; multi-class support vector machine; FPGA

1. Introduction

In research area of pattern recognition, because of multi-class support vector machines'(MC-SVM) reliability, generalization capacity, and proved performance superiority over other learning classification models such as multi-layer feedforward neural networks, it have enjoyed growing acceptance for application in all sorts of classification tasks. Introducing the posterior probability to the classification model can improve its performance[1]. So Platt et al. used the sigmoid function to map the output of SVM to posterior probability which enhanced the classify capability[2]. On that basis, Lin et al. proposed two improved algorithm[3] in which the second one had been proved more practical than other learning and classification algorithms[4]. Under the situation of needing real-time classification such as language recognition, FPGA implementation can be used to meet the requirement. Because the training of MC-SVM is time-consuming, its model parameter can be obtained off-line on the PC. So the main target of the FPGA implementation is how to fastly classify a input test set and output the corresponding posterior probability.

The FPGA-based SVM design proposal was first proposed by Anguita et al[5], in which the authors showed an efficient implementatjion of a kernel based perceptron in fixed-point digital hardware. Panadonikolakis et al proposed a scalable high-performance FPGA architecture of Gilbert's Algorithm on SVM, which maximally utilized the features of an FPGA device to accelerate the SVM training task for large-scale problems[6]. Irick et al presented an FPGA friendly implementation of a Gaussian Radial Basis SVM well suited for classification of grayscale images[7]. In this paper, we focus specifically on Lin's second improved classification algorithm, and then present its simplified algorithm which is suitable for FPGA implementation. We implement this algorithm pipeline based on Xilinx V5 and verify its performance on the language recognition system in which the real-time requirement is 10ms.

+ Corresponding author.
E-mail address: wishernzl@126.com

The rest of this paper is organized as follows: Section Ⅱ present a brief description of MC-SVM algorithm which is based on posterior probability. Section Ⅲ shows how to simplify the algorithm, providing a description of FPGA implement of the simplified algorithm which contrasts the computation complexity with the original algorithm. Section Ⅳ shows the experimental results of the algorithm applied to language recognition system and analysis on the results. In the end, the conclusion is provided.

2. Algorithm Theory Of Posteriroi Probability –Based MC-SVM

The process of the second improved classification algriothm in [3] can be divided into the following four steps(In this paper, × stands for the scalar product by corresponding elements of a vector or matrix, · stands for the dot product of vector or matrix and the C-based MC-SVM algorithm is based on LibSVM software[8]):

Step1: Given the observation X and the class label y, we normalize each dimenson of the X vector so as to accelerate the convergence and avoid the impact of sample outliers on SVM's correct classification. The formula read as follows:

$$x_n[idx] = lower + (upper - lower) \times \frac{x[idx] - \min[idx]}{\max[idx] - \min[idx]} \tag{1}$$

where we assume that X is m-dimension vector and lower=-1, upper=1(because the normalized range is [-1,1]). x[idx] is the idxth dimension of X, max[idx] is the maximum value of the idxth dimension in all of the training data and min[idx] is the minmum value of the idxth dimension among all of the training data. From the formula, we come to an conclusion that it needs m addition, 2m subtraction, m multiplication and m division to compute (1).

Step2: Compute f(x) is defined as the distance value from normalized sample vector X to the separating hyperplane. Because of the software LibSVM uitilize the "one against one" strategy to classify, there need n(n-1)/2 SVM classifiers. In this paper, we adopt linear kernel function.

$$f_i(\vec{x}) = \vec{\omega}_i^T \cdot \vec{x}_n + b_i \quad \text{i=1,2...,n(n-1)/2} \tag{2}$$

where $\vec{\omega}_i$, b_i is the normal vector and bias of the ith SVM classifier's separating hyperplane respectively. (2) needs m addition and m multiplication for one SVM output f(x). So it needs mn(n-1)/2 addition and mn(n-1)/2 multiplication in all.

Step3: Using sigmoid function to map f(x) to the estimated pairwise class probability r_{ij} of $\mu_{ij} = P\{y = i \mid y = i \ or \ j, x\}$:

$$\begin{aligned} r_{ij} = Sigmoid(f_i(\vec{x})) &= \frac{1}{1+\exp(A_i \times f_i(\vec{x}) + B_i)} \\ &= \frac{1}{1+\exp(x^{'})} \qquad \text{i} \neq \text{j} \end{aligned} \tag{3}$$

where A_i and B_i are the parameters of sigmoid function, they constitute vector $\vec{A}$ and $\vec{B}$. From (3), it is easy to know $r_{ij} = 1 - r_{ji}$, and with regard to one r_{ij}, it needs 2 addition, 1 multiplication, 1 division and 1 exponentation. So we require n(n-1) addition, n(n-1)/2 multiplication, n(n-1)/2 division and n(n-1)/2 exponentation to compute n(n-1)/2 r_{ij}.

Stpe4: Uitilize the pairwise coupling probability r_{ij} to solve the posterior probability vector P, which involve solving optimization problems. The optimization formulation read as follows:

$$\min_P \sum_{i=1}^{n} \sum_{j:j \neq i} (r_{ji} p_i - r_{ij} p_j)^2 \quad subject\ to \ \sum_{i=1}^{n} p_i = 1. \tag{4}$$

Note that p_i is the ith element of n-dimension vector P. The objective function of (4) can be rewritten as

$$\min_P 2P^T QP \qquad \text{thereinto } Q_{ij} = \begin{cases} \sum_{s \neq i} r_{si}^2 & i = j \\ -r_{ji} r_{ij} & i \neq j \end{cases} \tag{5}$$

It can be implementated by appendix D of [3] which is divided into two sections:

Section 1: Compute symmetric and semi-definite matrix Q which have n rows and n columns. It is easy to know that 3n(n-1)/2 multiplication is required to compute Q_{ij} and r_{si}^2, n(n-1) cumulative sum to obtain Q_{tt}.

Section 2: Using Q matrix and iteration algorithm to get posterior probability vector P. The pseudo-code of this iteration algorithm can be seen in the figure 4. Each class probability is 1/n at the initialization. The threshold of the stopping condition exps in the pseudo-code comes from experience value. If the iteration times is 100 but the stopping condition still has not been met, it will stop and output the final posterior probability. It is easy to analyse that the computation(one loop) is n^2+4n-1 addition, 2n subtraction, $2n^2$+4n multiplication and $2n^2$+2n division.

3. FPGA Design Of Posterior Probability-based MC-SVM Classifier

3.1. FPGA Design of Simplified Module and Sigmoid Function

From the algorithm flow we can find that (1) is equivalent to the following formula:

$$x_{scale}[idx] = \frac{2}{\max[idx]-\min[idx]} \cdot x[idx] + \frac{\max[idx]+\min[idx]}{\min[idx]-\max[idx]} \tag{6}$$

Its matrix form is $\vec{X}_{\text{scale}} = \vec{M} \times \vec{X} + \vec{N}$, where $\vec{M}$ is constituted by the multiplication item and $\vec{N}$ is constituted by the addition item. It has the same form with $\vec{\omega}_i^T \cdot \vec{x}_{scale} + b_i$ in (2) and $A_i \times f_i(\vec{x}) + B_i$ in (3). So it can be merged and form one matrix expression: $\vec{X}^{'} = \vec{A}^{'} \cdot f_i(\vec{x}) + \vec{B}^{'}$, where $\vec{A}^{'} = (\vec{A} \times (\vec{\omega}_i \times \vec{M}))^T$ and $\vec{B}^{'} = \vec{A} \times (\vec{\omega}_i^T \cdot \vec{N} + \vec{b}) + \vec{B}$. They can be pre-stored in ROM. Beacause (6) is only involved multiply accumulation, it needs mn(n-1) addition and mn(n-1) multiplication to compute a m-dimension input sample.

Sigmoid function is showed in figure 2. This non-linear function is odd-symmetry at (0,0.5) and its range is [0],[1]. In this paper, we use look-up table method to obtain pairwise coupling probability r_{ij} which is the output of sigmoid function. The value of sigmoid function in x=8 is 0.999667 and 3.3×10^{-4} in x=-8. When the input is bigger than 8, the value of the function is 1 and if the input is smaller than -8, its value is 0. In [0,8], it can be divided into 8 segments in which one can be divided into 256 segments equally. The 2048 segments need 11bit to addressing and 64Kbit ROM to store 32bit number. In (-8,0), the odd-symmetry can be used. Therefore, the mean absolute error of the look-up table is 3.4×10^{-4}. Its computation is ignored because of shifting and addressing are only involved in computing sigmoid function.

3.2. FGPA Design of Optimization Problem

1) *FPGA Design of Matrix Q Module*

Because of $Q_{ij} = \begin{cases} \sum_{s\neq i} r_{si}^2 & i=j \\ -r_{ji}r_{ij} & i\neq j \end{cases} = \begin{cases} \sum_{s\neq i} r_{si}^2 & i=j \\ r_{ij}^2 - r_{ij} & i\neq j \end{cases}$,

where r_{ij} and $r_{ij} = 1 - r_{ji}$ can be used to compute Q_{ij} $i \neq j$,and then r_{ij}^2 is obtained by $Q_{ij} + r_{ij} = r_{ij}^2$. Q_{tt} is computed by accumulating r_{ij}^2. It requires n(n-1)/2 multiplication to obtain Q_{ij}, n(n-1) addition to compute r_{ij}^2 and so is Q_{tt}.

2) *FPGA Design of Iteration Module*

The denominator of division operation on this module's data path is Q_{tt} or $1+\Delta$. That means it only needs 2 division and other division operation that can become multiplication so as to shorten the delay and reduce resource comsumption. During the processing of iteration, $Q_{tt} \to 0$ may leads to $\Delta \to \infty$, which can hardly be represented by fixed-point in FPGA implementation. But we can discover that the result of the iteration is $p_t \to 1, p_j \to 0,\ j \neq t$ when $\Delta \to \infty$ from the analysis of the inner loop in the iteration. Therefore, a threshold which can be obtained from the experience through the large data test is setted. When $\Delta >= \xi$, the iteration will stop and output $p_t = 1, p_j = 0,\ j \neq t$. Table Ⅰ show the contrast of each step between the original algorithm and simplified algorithm. We can discover that the latter operation is significantly decreased.

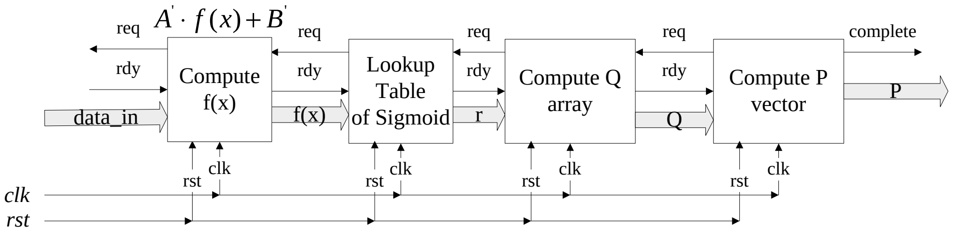

Fig.1 MC-SVM classification algorithm design diagram

TABLE I. THE OPERAND OF THE TWO ALGORITHM

	C-based floating point MC-SVM					FPGA-based MC-SVM		
	Step1	Step2	Step3	Step4		Step1	Step2	
				Q	Iteration		Q	Iteration
Addition	m	mn(n-1)/2	n(n-1)	n(n-1)	n^2+4n-1	mn(n-1)/2	3n(n-1)	$2n^2$+4n-1
Subtraction	2m	N/A	N/A	N/A	2n	N/A	N/A	2n
Multiplication	m	mn(n-1)/2	n(n-1)/2	3n(n-1)/2	$2n^2$+4n	mn(n-1)/2	n(n-1)/2	$4n^2$+4n
Division	m	N/A	n(n-1)/2	N/A	$2n^2$+2n	N/A	N/A	2n
Exp	N/A	N/A	n(n-1)/2	N/A	N/A	N/A	N/A	N/A

Figure 1 is the block diagram of the FPGA-based simplified algorithm. The data exchange is controled by handshake signals between modules. The input and output of each module use dual-port block RAM in order to achieve pipeline and the data path is controlled by finite state machine(FSM). Each module's clock and reset signals are synchronous. The signals is triggered by clock rising edge.

4. Language Recognition Experiments

The FPGA-based MC-SVM classifier is applied to language recognition experiment which the real-time requirement is 10ms. For feature extraction, MFCC and its SDCC constitute a total of 56 features per frame. Additional processing included RASTA, 0/1 feature noramlization and VTLN. For a language and gender independent GMM-UBM, we trained a 512 mixture GMM with 5 iterations of EM adapting parameters of the means. The SVM use the linear kernel function as described in section Ⅲ. The SVM input is the projection from the GMM supervector(GSV) which is formed by adaped model using MAP adaptation of GMM-UBM's means to anchor space which . That is a 246-dimension vector which is the target of the posterior probability-based MC-SVM's classification. The output of the MC-SVM is the maximun of the posterior probability vector and the corresponding language lable. The test speech utterances are 3 languages of the OGI-TS language 30s telephone speech corpus: 590 English, 228 Japanese and 435 Chinese. The performance of the FPGA-based MC-SVM was evaluated from these test data. The parameters of the MC-SVM based on posterior probability were trained with LibSVM. The threshold of the iteration module in this experiment is 1024.

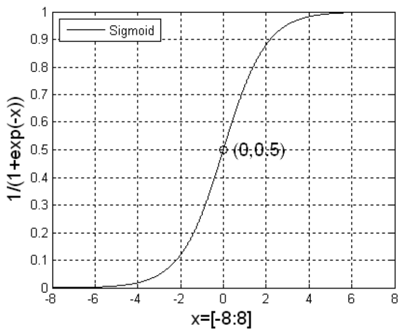

Fig.2. Simoid function

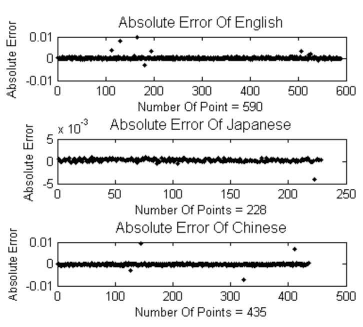

Fig.3. Absolute Error between the two algorithm

```
Iterative algorithm pseudo code
1. Input: $Q$ matrix, $p_i = \frac{1}{n}, eps = \frac{0.005}{n}, \max\_iter = 100, t = 0$ and $j = 0$;
2. while iter ≤ max_iter    do
3. {    $QP = Q \cdot P$;  $PQP = P^T \cdot QP$;
4.      if  $\max(|QP_t - PQP|) < eps$  then  break out;
5.      else
6.      {   while  t<n  do
7.          { calculation $\Delta = \frac{-QP_t + PQP}{Q_{tt}}$ ; update $p_t = p_t + \Delta$ ;
8.             update  $PQP = \frac{PQP + \Delta \cdot (\Delta \cdot Q_{tt} + 2 \cdot QP_t)}{(1+\Delta)^2}$;
9.             while  j<n  do
10.           { update $QP_j = \frac{QP_j + \Delta \cdot Q_{tj}}{1+\Delta}$ and $p_j = \frac{p_j}{1+\Delta}$;}
11.         }
12.     }
13. Output:  $\arg\max_i [p_i]$.
```

Fig.4 Pseudo-code of the iterative algorithm

TABLE II. RECOGNITION RATE OF FPGA-BASED ALGORITHM AND THE

	Floating point SVM		FPGA-based MC-SVM		floating point recognition rate	FPGA-based recognition rate
	test	correct	test	correct		
Chinese	435	423	435	422	97.4%	97.0%
Japanese	228	208	228	205	91.22%	89.92%
English	590	493	590	485	83.55%	82.2%

Figure 3 is the absolute error between the fixed-point output of FPGA-based simplified algorithm and the floating point output of the original algorithm which is based on C language. The mean absolute error of English, Japanese and Chinese is 2.533×10^{-4}, 2.5326×10^{-4} and 2.485×10^{-4} respectively. Table Ⅱ shows the recognition rate of the FPGA-based simplified algorithm and the original algorithm. From the table, we can conclude that the FPGA-based MC-SVM does not affect the recognition rate. In this experiment, m=246 and n=3. So we can obtain that the operand declined 66.3%. We obtain the time delay of FPGA-based the simplified algorithm is 0.7358ms from the simulation of ISE10.1 software, which meet the real-time requirement of the system.

5. Conclusion

This paper simplifies a posterior probabiltiy-based MC-SVM classification algorithm and propose a FPGA-based hardware architecture. The analysis results prove the operand of the simplified algorithm declined. Experimental results on language recognition analysis shows that the posterior probability based on FPGA implementation of the MC-SVM classifier hardly affect the performance but meet the system real-time requirement.

6. References

[1] Duda R, Hart Pe. Pattern Classification and Scne Analysis[M]. Wiley, 1973.

[2] J. Platt, Probabilistic outputs for support vector machines and comparison to regularized likehood methods[J]. In A.J. Smola, editors, Anvances in Large Margin Classifiers, Cambridge, MA, MIT Press, JUE. 2000, pp. 61-74.

[3] Ting-Fan Wu, Chi-Jen Lin, Ruby C Weng. Probability Estimates for Multi-class Classification by Pairwise Coupling[J]. Journal of Machine Learing Research, (5)2004, pp. 975-1005.

[4] Kai-Bo Duan and S. Sathiya Keerthi. Which is the Best Multiclass SVM Method-An Empirical Study[J]. LNCS 3541, 2005. pp. 278-285.

[5] D. Anguita, S. Ridella, S. Rovetta. Circuital implementation of supprot vector machines[J]. Electron Lett, 1998, 34(16): 1596-1597.

[6] M. Papadonikolakis and C-S. Bougains. Efficient FPGA mapping of Gibert’s algorithm for svm training on large-scael classification problems[J]. In FPL’08, 2008.

[7] K.M. Irick, M. DeBole et al. A Hardware Efficient Support Vector Machine Architecture for FPGA[J]. The 16th International Symposium on Field-Programmable Custom Computing Machines. pp. 304-305, 2008.

[8] C. Lin. LIBSVM: A library for support vector machines. http://www.csic.ntu.tw/cjlin/libsvm/index.html.2009.

[9] Koldo Basterrextea, Jose Manuel Tarela，Ines Del Campo. Approximation of Sigmoid Function and The Derivative for Artificial Neurous. 2000(151).1,pp.18-24.

[10] Elad Noor, Hagai A. Efficient Language Identification using Anchor Model and Support Vector Machines[J].pp.1-6. 2006.