Distributed Quadratic Programming Solver for Kernel SVM using Genetic Algorithm

Dinesh Singh and C. Krishna Mohan
Visual Learning and Intelligence Group (VIGIL),
Department of Computer Science and Engineering,
Indian Institute of Technology Hyderabad, India
Email: {cs14resch11003, ckm}@iith.ac.in

***Abstract*—Support vector machine (SVM) is a powerful tool for classification and regression problems, however, its time and space complexities make it unsuitable for large datasets. In this paper, we present GeneticSVM, an evolutionary computing based distributed approach to find optimal solution of quadratic programming (QP) for kernel support vector machine. In GeneticSVM, novel encoding method and crossover operation help in obtaining the better solution. In order to train a SVM from large datasets, we distribute the training task over the graphics processing units (GPUs) enabled cluster. It leverages the benefit of the GPUs for large matrix multiplication. The experiments show better performance in terms of classification accuracy as well as computational time on standard datasets like GISETTE, ADULT, etc.**

I. Introduction

The support vector machine (SVM) has been immensely successful in classification of diverse inputs from the fields of genomics, e-commerce, surveillance systems etc. However, to make intelligent decisions about such data is becoming increasingly difficult due to easy availability of high volume data [8], most noticeably in the form of text, images and videos. Since user preferences and trends keep on changing fluidly, analysis of such a large volume of data to support decision making is almost inevitable. Also, the support vector machine (SVM) has been used for classification and regression problem in many areas due to its generalization capabilities. Support vector machine (SVM) is based on statistical learning theory developed by Vapnik [3]. SVM solves the problem of over-fitting and makes a generalized model from the least number of samples. Several implementations of SVM are available, such as LIBSVM [2], LS-SVM [15], SVMlight [7]. However, the time and space complexities of SVM increase rapidly with increase in size of the training data. This makes training SVM difficult for large scale datasets. The time complexity for a standard SVM training is $O(n^3)$ and the space complexity is $O(n^2)$, where n is the size of training dataset [16]. It is thus computationally infeasible on very large data sets. The core of SVM is in solving a quadratic programming (QP), a NP hard problem which separates support vectors from the rest of the training data.

Sequential minimal optimization (SMO) is the state of the art QP solver which is used in LIBSVM, an implementation of SVM. But this method is sequential, so we can not leverage the benefit of high performance distributed computing environments like high performance cluster (HPC), cloud cluster, GPU cluster etc. Stochastic gradients decent (SGD) method can be distributed in order to train on large scale datasets. But this method works only for linear kernels. There is no existing true parallel or distributed algorithm to solve the constrained quadratic programming problem used to separate the support vectors from the training data for kernel SVM. In order to improve the training speed of SVM, many approaches have been proposed in the literature. These approaches can be categorized into *decomposition based approaches* and *partitioning based approaches*. The decomposition based approaches efficiently address the space complexity, however, time complexity remains a challenge. The partitioning based parallel and distributed SVM methods partition the data into smaller partitions and train SVM over them independently and later combine them to produce final support vectors. But, the partitioning based distributed SVM approaches [14], [1], [6], [20] are prone to loss of accuracy and high communication overhead.

In [5], Herrero-Lopez *et al.* accelerate SVM training by integrating graphics processing unit (GPU) into MapReduce clusters. It distributes the matrix multiplication tasks during the sequential update of the Lagrangian multipliers, however, it does not allow the desired level of acceleration due to the sequential nature of the SVM. The evolutionary computing shows success in order to find a solution near to the optimal solution quickly for NP hard problems and the computations are easy to perform independently in a distributed environment. Also, the execution of genetic algorithms can be accelerated by utilizing the massive parallelization power of the GPU cluster for training over large datasets. GPU-based parallel genetic algorithm are also proposed by [11][12][17][18] for various applications. Several researchers also use genetic algorithm for parameter tuning, i. e. selecting the best performing parameters for SVM training [19].

However, in this work, we aim to exploit the evolutionary computing based optimization ability of the genetic algorithm to perform distributed computation in finding the optimal solution for the SVM i. e. support vectors and their respective $\boldsymbol{\alpha}$ coefficients. Merz *et al.* [9] use genetic algorithm for binary quadratic programming (BQP) problem, but this is not applicable for the real valued QP problem in SVM. Herrera *et al.* [4] implement genetic algorithm based support vector regression.

It represents the real numbers into binary strings and apply traditional genetic algorithm. Also, it does not explore the automatic tuning of the various parameters used in kernel SVM like regularization parameter C, what is considered an open research area. In [13], Silva *et al.* implement least square SVM (LS-SVM) using genetic algorithm. The disadvantages of these methods are: 1) Sparsity is not incorporated, due to which all vectors in the training dataset become support vectors (SVs). 2) Generation of large number of invalid solutions reduce the computational efficiency. Apart form these limitations, all the above discussed methods use sequential computation only.

In this paper, we propose an evolutionary computing based quadratic programming (QP) solver for distributed training of kernel SVM known as GeneticSVM. The abilities of the proposed GeneticSVM are:

1) It represents candidate solutions for SVM using sequences of random real numbers called random key encoding, instead of commonly used binary coded string sequences. The crossover operation is also directly defined on the proposed random key encoding. The random key encoding reduces the computational time by avoiding decimal to binary and binary to decimal conversions when using binary encoded genetic algorithm.
2) It generates only valid candidate solutions during initial population generation and reproduction, instead of a large number of invalid solutions generated in binary encoded genetic algorithm.
3) For large matrix multiplication, it leverages the massively parallel computation power of GPUs.
4) It is suitable for training a SVM classifier from large datasets, because genetic algorithm can be distributed to any scale in various distributed computing environments. It presents two distributed frameworks for GPU enabled HPC or cloud cluster. First framework reduces the training time and achieves fast convergence. Second framework is for training from large datasets.

The rest of the paper is organized as follows: The proposed GeneticSVM is discussed in section II. Section III describes the experimental setup, evaluation method and results. We conclude in section IV with references at the end.

II. Proposed GeneticSVM

This section presents the proposed GeneticSVM for the optimization of quadratic programming (QP) problem for support vector machine (SVM).

Let $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ be the dataset with n feature vectors, $\mathbf{x}_i \in \mathbb{R}^d$ be the d dimensional feature vector and $y_i \in \{-1, +1\}$ be the class label. Then the QP problem for SVM is to maximize:

$$J(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T\mathbf{e} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha}, \tag{1}$$

where $q_{ij} = y_i y_j K(\mathbf{x}_i^T, \mathbf{x})$ and $\alpha_i \in \boldsymbol{\alpha}$ are the Lagrangian multipliers. A valid solution must satisfy following constraints:

$$\boldsymbol{\alpha}^T\mathbf{y} = 0, \tag{2}$$

$$0 \leq \alpha_i \leq C, \quad \forall \alpha_i \in \boldsymbol{\alpha}. \tag{3}$$

Here, C is a regularization parameter. Solving Equation (1) gives $\boldsymbol{\alpha}$ and the value of bias b. All non-zero $\alpha_i \in \boldsymbol{\alpha}$ are called support vectors. Let m be the number of support vectors. Then the decision of a vector $\mathbf{x}$ is predicted using support vectors and their corresponding α_i values using the following decision function:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i^T, \mathbf{x}) + b\right). \tag{4}$$

As discussed earlier, existing sequential minimal optimization (SMO) solves Equation (1) sequentially and also result in a solution that is not necessarily optimal. In the subsequent section, we propose a solver for Equation (1) using genetic algorithm in order to obtain the better solution. Also, we proposed a distributed framework which performs distributed computation over GPU enabled cluster in order to reduce the time for SVM training.

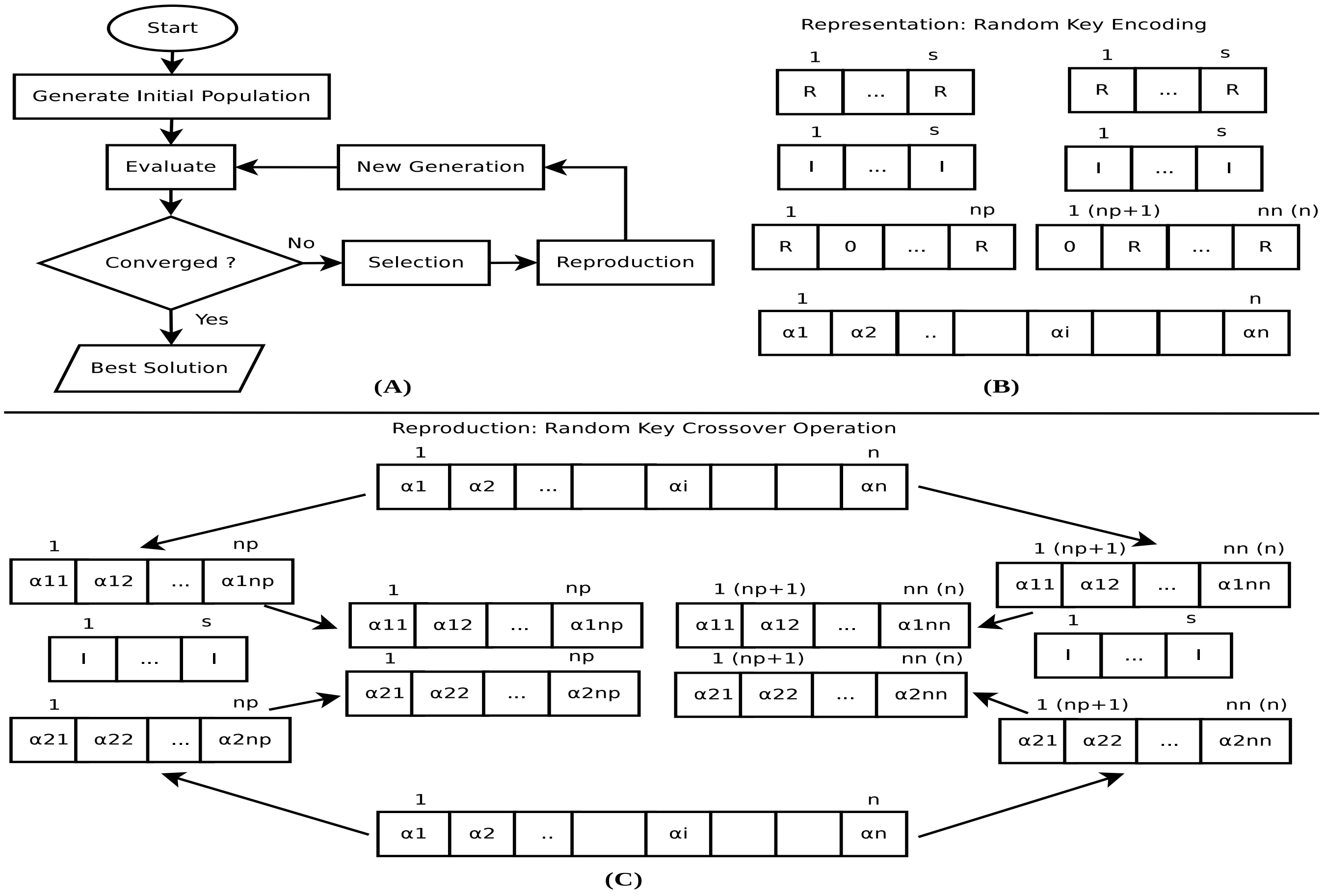
A. Proposed Genetic Algorithm based QP Solver

Here, we propose a genetic algorithm based solver for QP in Equation (1). The solution for Equation (1) is the optimal set of Lagrangian multipliers $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^n, \alpha_i \in \mathbb{R}$. As shown in Fig. 1, it generates random solutions i.e. $\boldsymbol{\alpha}_j$ and represents each solution using its n values of α_i, called random key representation. For evaluating the fitness of each solution, we use objective function in Equation (1) as fitness function. Reproduction operations are performed directly on the random keys of two candidate solution in order to generate new solutions. The details of the operations performed in proposed genetic algorithm based QP solver for searching the best solution is given here. As shown in Fig. 1-(A), the steps in a genetic algorithm include:

1) *Encoding:* The proposed approach uses random key encoding in order to represent the candidate solutions. The candidate solutions are the positive real valued $\boldsymbol{\alpha} \in \mathbb{R}^n$, where n is the number of vectors in the training set. The encoding should satisfy the constraints given in Equation (2) & (3). The Algorithm 1 generates $\boldsymbol{\alpha} \in \mathbb{R}^n$ which satisfies both the constraints. Let n_p be the number of positive class vectors and n_n be the negative class vectors. As shown in Fig. 1-(B), it generates two random vectors $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_n$ of size n_p and n_n with sparsity s_p and s_n, respectively. Hence, output vectors $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_n$ have only s_p and s_n non zero values, respectively. In order to satisfy constraints given in Equation (2) & (3), the vector $\boldsymbol{\alpha}_p$ and $\boldsymbol{\alpha}_n$ are normalized with factor f_p and f_n, respectively as follows:

$$\boldsymbol{\alpha}^p \leftarrow \boldsymbol{\alpha}^p \times f_p; \text{ where } f_p \leftarrow \frac{n \times C}{4 \times \mathbf{e}^T\boldsymbol{\alpha}^p}, \tag{5}$$

$$\boldsymbol{\alpha}^n \leftarrow \boldsymbol{\alpha}^n \times f_n; \text{ where } f_n \leftarrow \frac{n \times C}{4 \times \mathbf{e}^T\boldsymbol{\alpha}^n}. \tag{6}$$



Start
Generate Initial Population
Evaluate
New Generation
Converged ?
No
Selection
Reproduction
Yes
Best Solution
(A)
Representation: Random Key Encoding
(B)
Reproduction: Random Key Crossover Operation
(C)

Fig. 1. GeneticSVM operations. (A) The flow diagram of the steps in genetic algorithm. (B) The process of the solutions representation using random key encoding. (C) The process of crossover operation for reproduction of new candidate solutions.

Then the final solution is represented by $\boldsymbol{\alpha}$ as follows:

$$\boldsymbol{\alpha} \leftarrow [\boldsymbol{\alpha}^p, \boldsymbol{\alpha}^n]. \tag{7}$$

2) *Initial Population:* We generate initial population $\mathbf{A}$ of size m using Algorithm 2.

$$\mathbf{A} \leftarrow \{\boldsymbol{\alpha}_j\}_{j=1}^{m} \tag{8}$$

3) *Evaluation:* In order to evaluate the fitness of a solution $\boldsymbol{\alpha}$, the objective function $J(\boldsymbol{\alpha})$ in Equation (1) is used as the fitness function. The fitness value f_j for j^{th} solution $\boldsymbol{\alpha}_j$ is given by:

$$f_j = \boldsymbol{\alpha}_j^T \mathbf{e} - \frac{1}{2}\boldsymbol{\alpha}_j^T \mathbf{Q} \boldsymbol{\alpha}_j. \tag{9}$$

Equation (9) gives the fitness of single candidate solution only. In order to utilize the GPUs efficiently, we can calculate the fitness of all m, $\boldsymbol{\alpha}_j \in \mathbf{A}$ as:

$$\mathbf{f} \leftarrow \mathbf{A} \times \mathbf{e} - \frac{1}{2}((\mathbf{A} \times \mathbf{Q}).\mathbf{A}) \times \mathbf{e}. \tag{10}$$

4) *Selection:* For selection, *roulette wheel* selection is used, however other methods such as rank selection can also be used. The fitness value of each $\boldsymbol{\alpha}_j \in \mathbf{A}$ is used to associate a probability of selection. Let f_j be the fitness of $\boldsymbol{\alpha}_j$, then its probability of being selected is given by:

$$p_j = \frac{f_j}{\Sigma_{k=1}^{m} f_k}. \tag{11}$$

5) *Reproduction:* For reproduction, we use only crossover. The crossover operation is a random r-site crossover in which two parents generate four children. As shown in Fig. 1-(B), it randomly selects two solutions $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ from mating pool and separates them into $\boldsymbol{\alpha}_1^p$, $\boldsymbol{\alpha}_1^n$, $\boldsymbol{\alpha}_2^p$, and $\boldsymbol{\alpha}_2^n$. Random key crossover is applied separately on pairs i.e. $\boldsymbol{\alpha}_1^p$, $\boldsymbol{\alpha}_2^p$ and $\boldsymbol{\alpha}_1^n$, $\boldsymbol{\alpha}_2^n$. The random key crossover generates random integer indices $\boldsymbol{k}^p$ and $\boldsymbol{k}^n$ in the range 1 to n_p and 1 to n_n, respectively. And the values of $\boldsymbol{\alpha}_1^p$ and $\boldsymbol{\alpha}_1^n$ are exchanged with $\boldsymbol{\alpha}_2^p$ and $\boldsymbol{\alpha}_2^n$ at the respective indices in $\boldsymbol{k}^p$ and $\boldsymbol{k}^n$. However, $\boldsymbol{\alpha}_1^p$, $\boldsymbol{\alpha}_2^p$, $\boldsymbol{\alpha}_1^n$, $\boldsymbol{\alpha}_2^n$ may violate the constraint in Equation (2) due to exchange of values. So, in order to meet the constraint in Equation (2), the error i.e. the difference in the sum of values exchanged is calculated and adjusted. Then, we get the updated values of $\boldsymbol{\alpha}_1^p$, $\boldsymbol{\alpha}_2^p$, $\boldsymbol{\alpha}_1^n$, $\boldsymbol{\alpha}_2^n$ which

Algorithm 1 Random Key Encoding $genAlpha(n_p, n_n, d)$

```
Require: $n_p$ :Number of positive class samples in training dataset.
    $n_n$ :Number of negative class samples in training dataset.
    $d$ :Number of dimensions of sample vector.

1: $n \leftarrow n_p + n_n$;
2: $C \leftarrow rand\_int(d, 1)$; {random integer in the range 1 to $d$}
3: $s_p \leftarrow rand\_int(n_p, 1)$;
4: $\mathbf{r}^p \leftarrow rand\_int(n_p, s_p)$; {$s_p$ random integers in the range 1 to $n_p$}
5: $\boldsymbol{\alpha}^p_{\mathbf{r}} \leftarrow |\mathcal{N}(s_p \times 1)|$;
6: $f_p \leftarrow \frac{n \times C}{4 \times \mathbf{e}^T \boldsymbol{\alpha}^p}$;
7: $\boldsymbol{\alpha}^p \leftarrow \boldsymbol{\alpha}^p \times f_p$;
8: $s_n \leftarrow rand\_int(n_n, 1)$;
9: $\mathbf{r}^n \leftarrow rand\_int(n_n, s_n)$;
10: $\boldsymbol{\alpha}^n \leftarrow |\mathcal{N}(s_n \times 1)|$;
11: $f_n \leftarrow \frac{n \times C}{4 \times \mathbf{e}^T \boldsymbol{\alpha}^n}$;
12: $\boldsymbol{\alpha}^n \leftarrow \boldsymbol{\alpha}^n \times f_n$;
13: $\boldsymbol{\alpha} \leftarrow [\boldsymbol{\alpha}^p, \boldsymbol{\alpha}^n]$;
14: return $\boldsymbol{\alpha}$;
```

Algorithm 2 Initial Population Generation

```
Require: $m$ :Size of the initial population
    $n_p$ :Number of positive class samples in training dataset.
    $n_n$ :Number of negative class samples in training dataset.
    $d$ : Number of dimensions of sample vector.

1: Initialize $A[m]$;
2: for $j = 1 \rightarrow m$ { in parallel} do
3:     $\boldsymbol{\alpha}_j \leftarrow genAlpha(n_p, n_n, d)$;{using Algorithm-1}
4:     $A[j] \leftarrow \boldsymbol{\alpha}_j$;
5: end for
6: return $A$;
```

will result into four new solutions:

$$\mathbf{c}_1 = [\boldsymbol{\alpha}^p_1, \boldsymbol{\alpha}^n_1], \tag{12}$$
$$\mathbf{c}_2 = [\boldsymbol{\alpha}^p_1, \boldsymbol{\alpha}^n_2], \tag{13}$$
$$\mathbf{c}_3 = [\boldsymbol{\alpha}^p_2, \boldsymbol{\alpha}^n_1], \tag{14}$$
$$\mathbf{c}_4 = [\boldsymbol{\alpha}^p_2, \boldsymbol{\alpha}^n_2]. \tag{15}$$

The complete procedure of the new solution generation using random r-site crossover is given in Algorithm 3.

6) *Elitism:* Lets us consider the initial population size $m = 100$. Then, the population at $(g+1)^{th}$ generation retains 4-best solutions from g^{th} generation. And 92 new solutions are reproduced using $23(= 92/4)$ crossover operations using Equation (3) and the remaining 4 are the new solutions generated using Algorithm 1 as generated in the initial population.

The proposed GeneticSVM solves the QP problem in Equation (1) with results comparable to SMO. However, time taken on a single processor is too high. In order to reduce the training time, we perform distributed computations in cloud environment as presented in the next section.

Algorithm 3 Random r-Site Crossover

```
Require: $\boldsymbol{\alpha}_1$ :First Parent.
    $\boldsymbol{\alpha}_2$ :Second Parent.
    $n_p$ :Number of positive class samples in training dataset.
    $n_n$ :Number of negative class samples in training dataset.

1: $r \leftarrow rand\_int(n_p, 1)$;
2: $\mathbf{k}^p \leftarrow rand\_int(n_p, r)$;
3: $\boldsymbol{\alpha}^p_1 \leftarrow \{\alpha^p_{1i}\}^{n_p}_{i=1}$;
4: $\boldsymbol{\alpha}^p_2 \leftarrow \{\alpha^p_{2i}\}^{n_p}_{i=1}$;
5: $\mathbf{t}^p_1 \leftarrow \{\alpha^p_{1i}\}^{k^p_r}_{i=k^p_1}$;
6: $\mathbf{t}^p_2 \leftarrow \{\alpha^p_{2i}\}^{k^p_r}_{i=k^p_1}$;
7: $\{\alpha^p_{1k^p_i} \leftarrow t^p_{2i}\}^r_{i=1}$;
8: $\{\alpha^p_{2k^p_i} \leftarrow t^p_{1i}\}^r_{i=1}$;
9: $\epsilon \leftarrow \frac{\mathbf{e}^T \mathbf{t}^p_1 - \mathbf{e}^T \mathbf{t}^p_2}{2}$
10: if $\epsilon > 0$ then
11:     $l \leftarrow rand\_int(n_p, 1)$
12:     $\alpha^p_{1l} \leftarrow \alpha^p_{1l} + \epsilon$
13:     while $\epsilon \neq 0$ do
14:         $l \leftarrow rand\_int(n_p, 1)$
15:         if $\alpha^p_{2l} \geq \epsilon$ then
16:             $\alpha^p_{2l} \leftarrow \alpha^p_{2l} - \epsilon$;
17:             $\epsilon = 0$;
18:         else
19:             $\alpha^p_{2l} \leftarrow 0$;
20:             $\epsilon \leftarrow \epsilon - \alpha^p_{2l}$
21:         end if
22:     end while
23: else
24:     $\epsilon \leftarrow |\epsilon|$
25:     $l \leftarrow rand\_int(n_p, 1)$
26:     $\alpha^p_{2l} \leftarrow \alpha^p_{2l} + \epsilon$
27:     while $\epsilon \neq 0$ do
28:         $l \leftarrow rand\_int(n_p, 1)$
29:         if $\alpha^p_{1l} \geq \epsilon$ then
30:             $\alpha^p_{1l} \leftarrow \alpha^p_{1l} - \epsilon$;
31:             $\epsilon = 0$;
32:         else
33:             $\epsilon \leftarrow \epsilon - \alpha^p_{1l}$
34:             $\alpha^p_{1l} \leftarrow 0$;
35:         end if
36:     end while
37: end if
38: Similarly calculate $\boldsymbol{\alpha}^n_1$ and $\boldsymbol{\alpha}^n_2$.
39: $\mathbf{c}_1 = [\boldsymbol{\alpha}^p_1, \boldsymbol{\alpha}^n_1]$;
40: $\mathbf{c}_2 = [\boldsymbol{\alpha}^p_1, \boldsymbol{\alpha}^n_2]$;
41: $\mathbf{c}_3 = [\boldsymbol{\alpha}^p_2, \boldsymbol{\alpha}^n_1]$;
42: $\mathbf{c}_4 = [\boldsymbol{\alpha}^p_2, \boldsymbol{\alpha}^n_2]$;
43: return $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4\}$;
```

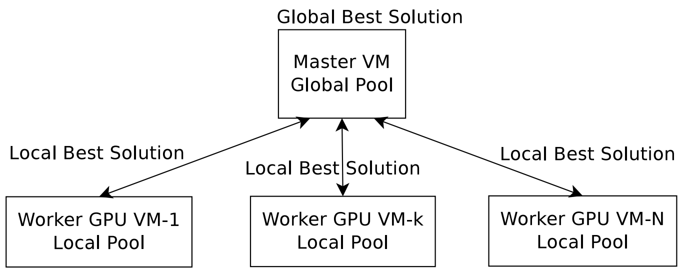


Fig. 2. Proposed architecture of Distributed GeneticSVM

B. Distributed Computing in Cloud Environment

The proposed genetic algorithm based QP solver is able to get the best solution for Equation (1) but the time taken is too high. However, unlike sequential minimal optimization (SMO), the proposed solver is easy to distribute. For GeneticSVM, we can utilize distributed environments like GPU enabled HPC or cloud clusters etc. Here, we present two distributed frameworks for GeneticSVM over the GPU enabled cloud cluster. The proposed frameworks work according to the available resources and size of the dataset. The first distributed GeneticSVM framework, run multiple instances of the algorithm and share the best solution among each others in order to achieve fast convergence. The second framework further distribute the task of a single instance of the algorithm for a large dataset.

1) Distributed GeneticSVM: The first framework is applicable when one virtual machine (VM) is able to store the data in physical memory but training time is too high. Here, we are considering availability of virtual resources provisioned over cloud environment. As shown in Fig. 2, we launch multiple instance of the GPU enabled virtual machines (VMs). One VM acts as master VM and all others act as worker VMs. Here, we maintain a global pool, $\mathbf{A}_G = \{\boldsymbol{\alpha}_{(k)}\}_{k=1}^{N}$ at master VM and a local pool $\mathbf{A}_L^k = \{\boldsymbol{\alpha}_j\}_{j=1}^{m}$ at k^{th} worker VM, $k = 1, 2, ..., N$. The kernel matrix $\mathbf{Q}$ is copied to all the worker VMs. Each worker VM generates the initial population, then do the fitness evaluation and send the best solution to the master VM. Master VM collects all the local solutions in the global pool $\mathbf{A}_G$, then it selects the global solution from the local solutions, and then broadcasts the best solution to all worker VMs. Further, each worker VM prepares the next generation which consists of global best solution, local best solution (if not winner worker VM), reproduced children solutions from previous generation solutions, and randomly generated solutions. This process is repeated until convergence. The fitness value (f) is calculated using Equation 10. Also, in this process, the N worker VMs send only best solution, thus, total N messages are passed over the network after each generation. This leads to very low communication which is of the order of $O(N)$. The sharing of best solutions leads to fast convergence.

2) Distributed GeneticSVM for Large Dataset: The first distributed framework i.e. *Distributed GeneticSVM* is not applicable for large dataset. Because the size of kernel matrix increases quadratically $O(n^2)$ with an increase in dataset size n. Thus, for a large dataset it is not an efficient way to store kernel matrix in one worker VM and execute the task. Thus in this framework for distributed GeneticSVM, we distribute the kernel matrix $\mathbf{Q}$ into L sub-worker VMs with GPU support, while worker VMs do not require GPU support as shown in Fig. 3. Each sub-worker VM with identifier $l = 1, 2, ..., L$ contains $\mathbf{Q}^l = \{\{q_{ij}\}_{i=1}^{n}\}_{j=\frac{(l-1)n}{L}}^{\frac{ln}{L}}$, a part of kernel matrix $\mathbf{Q}$, having size $n \times \frac{n}{L}$. The partial fitness $\mathbf{f}^l$ is calculated at each sub-worker machine as follow:

$$\mathbf{P} \leftarrow \mathbf{A} \times \mathbf{Q}^l, \tag{16}$$

$$\mathbf{A}^l \leftarrow \{\{a_{ij}\}_{i=1}^{m}\}_{j=\frac{(l-1)n}{L}}^{\frac{ln}{L}}, \tag{17}$$

$$\mathbf{P} \leftarrow \mathbf{P}.\mathbf{A}^l, \tag{18}$$

$$\mathbf{f}^l \leftarrow (\mathbf{A}^l \times \mathbf{e} + \frac{1}{2}\mathbf{P} \times \mathbf{e}). \tag{19}$$

Finally, the fitness value $\mathbf{f}$ is calculated as:

$$\mathbf{f} \leftarrow \sum_{l=1}^{L} \mathbf{f}^l. \tag{20}$$

III. Experiments and Results

The genetic algorithm is implemented in C/C++, CUDA, and OpenMPI over a GPU cluster running Ubuntu 14.04. The cluster contains two machines with specifications: 1) First machine has 2 Intel Xeon processors with 12 core each, 64GB physical memory and 6 Nvidia Tesla K20Xm GPUs with 5GB device memory each. 2) Second machine has 2 Intel Xeon processors with 24 core each, 128GB physical memory, 2 Nvidia Tesla K20c GPUs with 6GB device memory each. The large matrix multiplications are accelerated using GPUs. We have also successfully tested GeneticSVM on HPC with 512 nodes and on the Amazon Elastic Compute Cloud (EC2) using StarCluster [10]. StarCluster is a tool for dynamically creating, managing cluster on Amazon EC2 for testing MPI programs. Table I provides the details of the datasets used in the experiments.

TABLE I
DETAILS OF DATASETS USED TO EVALUATE THE PERFORMANCE OF GENETICSVM

Dataset	Dimensions	Training Size	Test Size
GISETTE	5000	6000	1000
ADULT (A1A)	123	1605	30956
ADULT (A2A)	123	2265	30296
ADULT (A3A)	123	3185	29376
ADULT (A4A)	123	4781	27780
ADULT (A5A)	123	6414	26147
ADULT (A6A)	123	11220	21341
ADULT (A7A)	123	16100	16461
ADULT (A8A)	123	22696	9865
ADULT (A9A)	123	32561	16281
MUSHROOMS	112	5000	3124
SVMGUIDE1	4	3089	4000

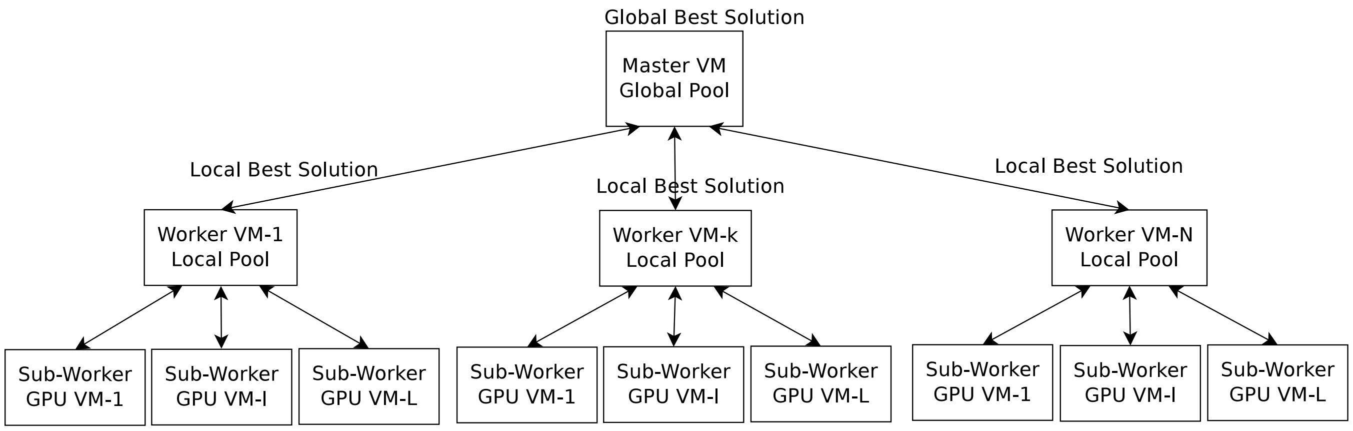


Fig. 3. Proposed architecture of distributed GeneticSVM for large dataset

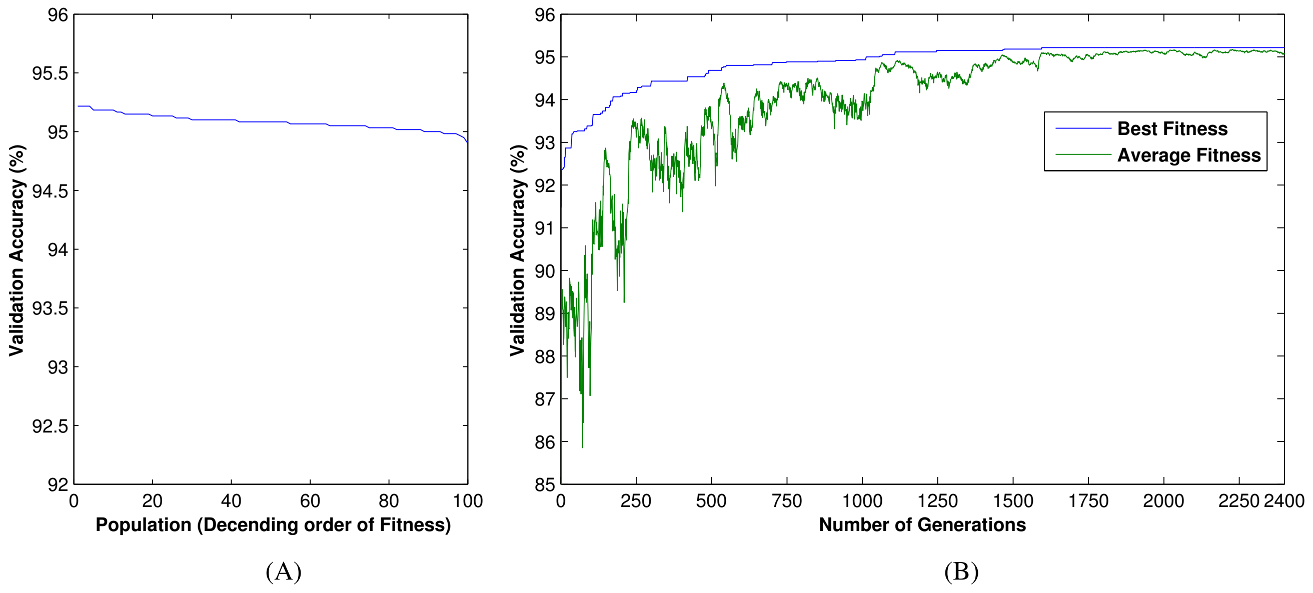


Fig. 4. Performance of classification for GeneticSVM on the GISETTE dataset after 2400 epoch. (A) All the candidate solution in the pool shows the high classification accuracy with a very low difference in between best and wrest solution. (B) Difference of best and average fitness reduces over the generations.

The results in Fig. 4-(A) show the fitness values of candidates in the pool after 2400 epochs and the results in Fig. 4-(B) reflect the improvement to the fitness index over the epochs. The presented experiment is conducted on the GISETTE dataset of OCR published during a NIPS challenge. The results show that the classification performance of the proposed approach is very close to sequential SVM. The results in Table II show the good classification ability of the proposed algorithm with a negligible loss of accuracy which can be further reduced by running algorithm for more number of generations. Fig. 5 shows the performance of classification while running the GeneticSVM algorithm multiple times. The results show very low standard deviations when running 10 times. Also, Fig. 5 shows that the proposed approach obtains the significant improvements in first few hundred iterations only, which shows the suitability of the encoding method and crossover operations used for generating new solutions. Fig. 6 shows the time taken by 100 worker VMs. Finally, when running the complete pipeline of the algorithm on various datasets, the GeneticSVM algorithm performs approximately 10-20 times faster than the LIBSVM as shown in Table III.

The proposed GeneticSVM performs better than existing partitioning based distributed SVMs approaches in terms of classification accuracy and time taken in training a SVM model. The proposed approach successfully achieves a comparable accuracy to sequential SVM for GISETTE dataset. Along with improvement in accuracy, proposed approach also performs approximately 3 times faster than the approach by You *et al.* [20]. Also, it can be observed that the loss of accuracy is less than 0.9% on other datasets, which demonstrates the efficacy of the proposed approach.

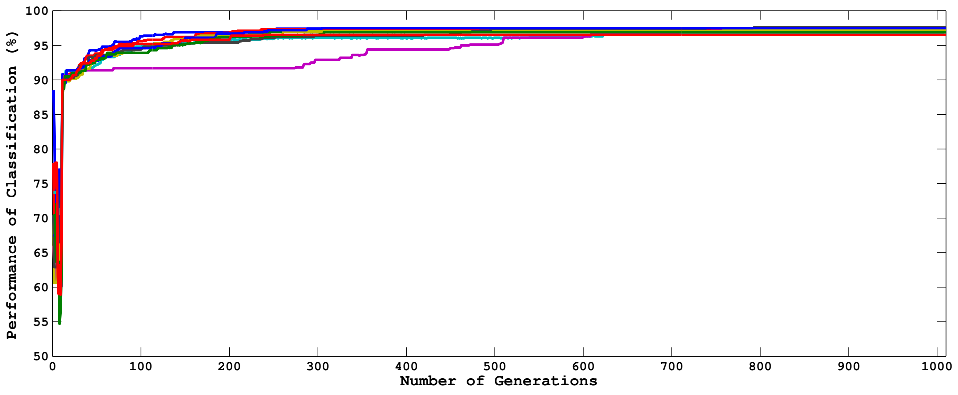


Fig. 5. Performance of genetic algorithm based optimization of QP problem for 10 runs using population size 1000 and pool size 2000 at each slave process and using population size 100 and pool size 1000 at master process.

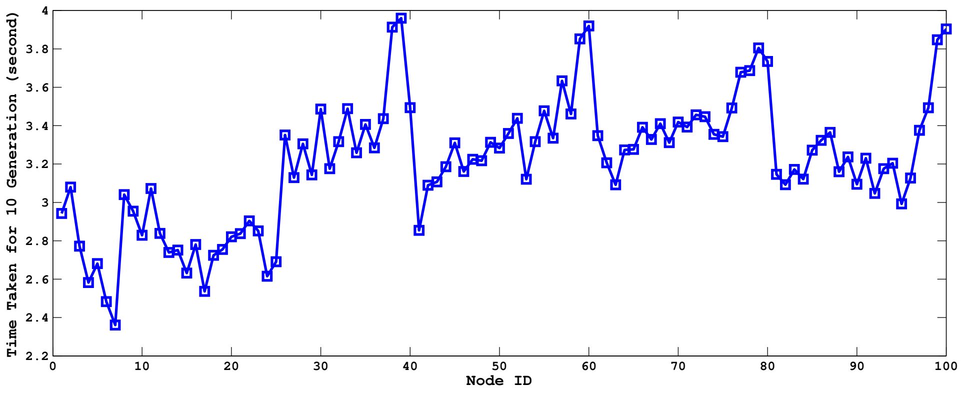


Fig. 6. Time taken by each process for 10 generations, each for population size 1000 and pool size 2000 at work VMs.

TABLE II
PERFORMANCE OF CLASSIFICATION (%) OF THE GENETICSVM AND COMPARISON WITH SMO USING LIBSVM

DataSet Used	SMO **Accuracy (%)**	**GeneticSVM Loss (%)**	**Accuracy Accuracy (%)**
GISETTE	97.60	97.60	0.0
ADULT (A1A)	83.59	83.19	-0.4
ADULT (A2A)	83.98	83.28	-0.7
ADULT (A3A)	83.84	83.54	-0.3
ADULT (A4A)	83.96	83.26	-0.7
ADULT (A5A)	84.17	83.37	-0.8
ADULT (A6A)	84.17	83.27	-0.9
ADULT (A7A)	84.58	83.78	-0.8
ADULT (A8A)	85.01	84.31	-0.7
ADULT (A9A)	84.82	84.52	-0.3
MUSHROOMS	97.09	96.39	-0.7
SVMGUIDE1	66.93	66.33	-0.6

TABLE III
TRAINING TIME (SECONDS) OF THE GENETICSVM AND COMPARISON WITH SMO USING LIBSVM

DataSet Used	**SMO** (Seconds)	**GeneticSVM (Mean±Var.) (Seconds)**	**Scaling**
GISETTE	214	9.2091 ± 1.3368	$\approx 20\times$
ADULT (A7A)	11.84	0.8013 ± 0.1307	$\approx 15\times$
ADULT (A8A)	22.97	1.4556 ± 0.2023	$\approx 15\times$
ADULT (A9A)	45.85	2.5473 ± 0.7359	$\approx 18\times$

IV. CONCLUSION

The partitioning based distributed SVMs have generally proven to be faster than sequential SVMs on large datasets. However, classification performance still lags behind. In the proposed GeneticSVM, we aimed at providing a distributed SVM approach which retains or improves the classification performance of sequential SVM while having the computational time gains as of distributed approachs on a large dataset. The GeneticSVM shows success in order to find the better solution quickly also the computations are efficiently distributed over GPU cloud cluster to leverage the benefit of the GPUs for large matrix multiplication. The experiments show better performance in terms of classification accuracy as well as computational time.

ACKNOWLEDGMENT

Supported by Microsoft Research India Travel Grant

REFERENCES

[1] N. K. Alham, M. Li, S. Hammoud, Y. Liu, and M. Ponraj, “A distributed SVM for image annotation,” in *Proc. of Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD)*, Yantai, Shandong, 10-12 Aug 2010, pp. 2983–2987.

[2] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Trans. on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[3] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[4] O. Herrera and A. Kuri, “An approach to support vector regression with genetic algorithms,” in *Proc. of the Fifth Mexican Int. Conf. on Artificial Intelligence, MICAI*, 2006, pp. 178–186.

[5] S. Herrero-Lopez, “Accelerating SVMs by integrating GPUs into MapReduce clusters,” in *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, 2011, pp. 1298–1305.

[6] C.-J. Hsieh, S. Si, and I. Dhillon, “A Divide-and-Conquer Solver for Kernel Support Vector Machines,” in *Proc. of Int. Conf. on Machine Learning*, vol. 32, no. 1, 2014, pp. 566–574.

[7] T. Joachims, “Making large-Scale {SVM} Learning Practical,” in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, ch. 11, pp. 169–184.

[8] X. Ke, R. Jin, X. Xie, and J. Cao, “A Distributed SVM Method based on the Iterative MapReduce,” in *Proc. of IEEE Int. Conf. on Semantic Computing (ICSC)*, no. 4, 2015, pp. 7–10.

[9] P. Merz and B. Freisleben, “Genetic algorithms for binary quadratic programming,” in *Proc. of the Genetic and Evolutionary Computation Conference*, 1999, pp. 417–424.

[10] MIT, “StarCluster.” [Online]. Available: http://star.mit.edu/cluster/index.html

[11] A. Munawar, M. Wahib, M. Munetomo, and K. Akama, “Advanced genetic algorithm to solve MINLP problems over GPU,” in *Proc. of IEEE Congress of Evolutionary Computation (IEEE CEC)*, 2011, pp. 318–325.

[12] M. Oiso, T. Yasuda, K. Ohkura, and Y. Matumura, “Accelerating steady-state genetic algorithms based on CUDA architecture,” in *Proc. of IEEE Congress of Evolutionary Computation (IEEE CEC)*, 2011, pp. 687–692.

[13] J. P. Silva and A. R. d. R. Neto, “Sparse Least Squares Support Vector Machines via Genetic Algorithms,” in *BRICS Congress on Computational Intelligence*, 2013, pp. 248–253. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6855857

[14] Z. Sun and G. Fox, “Study on Parallel SVM Based on MapReduce,” in *Proc. of Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2012, pp. 16–19.

[15] J. A. K. Suykens and J. Vandewalle, “Least Squares Support Vector Machine Classifiers,” *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[16] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, “Core VectorMachines: Fast SVMTraining on Very Large Data Sets,” *Journal of Machine Learning Research*, vol. 33, no. 2, pp. 211–220, 2008.

[17] M. Wahib, A. Munawar, M. Munetomo, and K. Akama, “Optimization of Parallel Genetic Algorithms for nVidia GPUs,” in *Proc. of IEEE Congress of Evolutionary Computation (IEEE CEC)*, 2011, pp. 803–811.

[18] K. Wang and Z. Shen, “A GPU-Based Parallel Genetic Algorithm for Generating Daily Activity Plans,” *IEEE Trans. on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1474–1480, 2012.

[19] C.-H. Wu, Y. Ken, and T. Huang, “Patent classification system using a new hybrid genetic algorithm support vector machine,” *Applied Soft Computing*, vol. 10, no. 4, pp. 1164–1177, 2010.

[20] Y. You, J. Demmel, K. Czechowski, L. Song, and R. Vuduc, “CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems,” in *Proc. of IEEE International Parallel and Distributed Processing Symposium*, Hyderabad, India, 2015, pp. 847–859.