Empirical Study of Time Efficiency and Accuracy of Support Vector Machines Using an Improved Version of PSVM

S. Tavara[12], H. Sundell[1], and A. Dahlbom[2]
[1]Department of Information Technology, University of Borås, Borås, Sweden
[2]School of Informatics, University of Skövde, Skövde, Sweden

***Abstract*—We present a significantly improved implementation of a parallel SVM algorithm (PSVM) together with a comprehensive experimental study. Support Vector Machines (SVM) is one of the most well-known machine learning classification techniques. PSVM employs the Interior Point Method, which is a solver used for SVM problems that has a high potential of parallelism. We improve PSVM regarding its structure and memory management for contemporary processor architectures. We perform a number of experiments and study the impact of the reduced column size p and other important parameters as C and γ on the class-prediction accuracy and training time. The experimental results show that there exists a threshold between the number of computational cores and the training time, and that choosing an appropriate value of p effects the choice of the C and γ parameters as well as the accuracy.**

Keywords: Parallel SVM; Processor Technology; Training Time

I. Introduction

High Performance Computing (HPC) tools are promising for improving performance in respect of time efficiency. As more computational power can be spent on less time, this enables the accuracy of results to be improved as well. The importance of utilizing HPC tools has been growing and parallel computing as the underlying method of HPC plays an important role for improving the time efficiency. Message Passing Interface (MPI) is one of the well-known parallel library standards that was originally designed for distributed memory systems, although it can handle shared and hybrid (combination of shared and distributed) memory architectures as well. The advantages of using the MPI library standard is well-known, albeit the efficiency of the parallel processing can be degraded due to data dependency, memory bandwidth, synchronization and communication bottlenecks.

Digital data is growing exponentially and hence analysis and calculation processes regarding big data are becoming computationally expensive. Within this context, machine learning is one of the fields that can take advantage of using HPC tools for improving the performance. Support Vector Machine (SVM) [17] is one of the classification machine learning techniques that has a wide area of applications and has got considerable attention during the last decade. The SVM problem is set up as a minimization problem and can thereafter be solved using classical optimization algorithms. Interior Point Method (IPM) [18] is a popular choice thanks to the high degree of parallelism inherent in it. However, IPM requires computing the inverse of a matrix which is computationally expensive. Besides, in most of cases the coefficient matrix derived from the system is ill-conditioned, meaning that the matrix is either singular or close to singularity which makes the problem computationally unstable. Therefore approximation and preconditioning methods are applied to prevent the ill-conditioning and to reduce the computational costs.

Cholesky Factorization (CF) [20] is one of the techniques that is used for achieving stable numerical solutions. CF factorizes matrix $A \in R^{n \times n}$ into a lower triangular matrix, i.e., $A = LL^T$, where $L \in R^{n \times n}$. Incomplete Cholesky Factorization (ICF) [20] is a truncated form of CF, i.e., $A = \hat{L}\hat{L}^T$, where $\hat{L}$ is a $n \times p$ sparse lower triangular matrix close to L, where p is the rank of $\hat{L}$. In ICF approximation, only p column vectors are calculated which makes this approximation quick and economical to compute since $p \ll n$. However, calculating the appropriate column rank value, p, is non-trivial. A lower value of p degrades the accuracy and a higher value of p increases the computational time. In this paper, we study the trade-off between the class-prediction accuracy and time efficiency for different p settings and different kernel functions. Furthermore, we study the correlation between the choice of p and the hyperparameters C and the γ value, in respect to the effect of the Gaussian and Laplacian kernels on the class-prediction accuracy and the training time.

The advantage of using distributed parallelism as MPI on SVM problems is well-known, although how to choose the appropriate numbers of computational cores is still unclear and non-trivial. In theory, increasing the number of machines from 1 to 10 will enhance the time efficiency 10 times, although this is way too idealistic. The reason for that is due to data communication, memory bandwidth and synchronization between different machines. In this paper, we investigate when the data communication part overtakes the parallel computation part in SVM, i.e., when increasing the number of cores no longer is beneficial. As Chang et al. [20] mention in their paper, due to communication and synchronization overheads, the speed-up is not linearly increased by increasing the number of cores, and after a specific number of cores the time efficiency even degrades.

In this respect, it is interesting to study the existence of a threshold that can give an idea of the appropriate number of cores. We theoretically show that there exists a threshold that suggests a minimum number of computational cores, while the maximum number of cores depends on the machines used.

In the following sections, we briefly describe SVM using PSVM software. We mention important processor technologies and then describe the preparation for experiments including the complexity analysis of SVM algorithm using PSVM. We continue with the experiments and the results and finally we discuss the obtained results.

II. Support Vector Machines

The basic idea in SVM is to search for a bipartite hyperplane that has a furthest possible distance to the closest training data points from both sides of the hyperplane. In order to find the optimal bipartite hyperplane, a simple SVM problem can be formulated as an primal quadratic optimization problem as following,

$$\begin{aligned} min \quad & P(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi \\ s.t. \quad & y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad , i = 1, 2, ..., n \\ & \xi_i > 0 \qquad\qquad\qquad\qquad , i = 1, 2, ..., n \end{aligned} \tag{1}$$

Where $\mathbf{w}$ is the weight vector for the hyperplane, $\mathbf{x}$ is the vector of observations, y_i is the class label and $y_i \in \{+1, -1\}$, b is the bias parameter, $\Phi(\mathbf{x})$ is the map function that maps the input vector $\mathbf{x}$ to the feature space, ξ_i is a classification error for sample i, and C is the parameter that makes a balance between the classification error and maximum margin.

With the help of Lagrangian multipliers, the primal optimization problem (1) can be reformulated into another optimization problem called dual optimization problem. Lagrangian multipliers relax the constraints of the primal optimization problem and reformulates the problem into a new quadratic optimization problem with simpler constraints as follows:

$$\begin{aligned} min \quad & D(\alpha) = \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{1}^T\boldsymbol{\alpha} \\ s.t. \quad & \sum_{i=1}^{n} y_i\alpha_i = 0 \qquad , i = 1, 2, ..., n \\ & 0 \leq \alpha_i \leq C \quad , i = 1, 2, ..., n \end{aligned} \tag{2}$$

Where $\boldsymbol{\alpha}$ is Lagrangian multipliers and $\alpha_i \in \boldsymbol{\alpha}$, $\mathbf{1}^T$ is a vector of ones and Q is a matrix of size $n \times n$ where $Q_{ij} = y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$. Equation (2) is known as dual equation. We reformulate Q_{ij} as $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)$ is called a kernel function. The advantage of using kernel function is that we can compute Q with out knowing the map function $\Phi(.)$ explicitly and instead we can choose an appropriate kernel function to calculate Q. Four well-known kernel functions are as follows:

- Linear kernel that is an inner product or dot product of input vector and is used for linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$,
- Gaussian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$,
- Laplacian kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma|\mathbf{x}_i - \mathbf{x}_j|)$,
- Polynomial kernel that is used for non-linear classification, i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + const)^d$, where $const = 0$ for homogeneous polynomial kernels and $const = 1$ for inhomogeneous kernels.

Different mathematical solvers are utilized to solve the primal, the dual, or the primal-dual optimization problem. Interior Point Method (IPM) starts from an initial point located in the interior feasible region and moves towards the optimal point(s) in an iterative manner. One of the advantages of IPM is its high degree of inherent parallelism compared to other solvers. IPM can use approximation methods such as Incomplete Cholesky Factorization (ICF) to approximate the original matrix $Q_{n\times n}$ to a smaller matrix as $H_{n\times p}$, where $p \ll n$. This approach can improve the time efficiency of the computations. In this paper, we have chosen the PSVM software that solves the SVM problem by utilizing IPM solver.

A. Parallel Support Vector Machines

We have conducted our experiments using the Parallelizing Support Vector Machines (PSVM) software that is originally written by Chang et al. [20]. PSVM uses ICF to reduce the problem size and IPM to solve the primal-dual optimization problem (1) and (2). ICF approximates the original $n \times n$ linear system Q to the smaller $n \times p$ linear system H, i.e., $Q \approx H^T H$ where n is the number of samples or instances, p is the reduced column size and $p \ll n$. The parallel ICF (PICF) is computed by a row-based round-robin algorithm and distributed evenly to the machines. The primal-dual problem is then solved by a parallel implementation of IPM and makes use of PICF. All the parallelization is done by utilizing the MPI library standard. Chang et al. claim that based on their empirical results when the reduced column size p is chosen as $\sqrt{n}$, then the error ϵ is negligible, where $trace(Q - H^T H) < \epsilon$. On the one hand they show the class-prediction accuracy obtained with some different values of p smaller than $\sqrt{n}$ in table 1 in their paper [20], albeit on the other hand they do not discuss further the impact of varying the p value for different kernel functions in a SVM problem. Therefore we further study the impact of different value of p on the class-prediction accuracy.

III. Related Works

SVM problems has got considerable attention in the last decade and the optimization problem derived from SVM problems are well studied. Challenges in SVM problems can be mentioned as the need for a large amount of memory for training samples [8][19] and the intense training time [1] when the problem size is large. This gives a motivation to use optimization methods along with HPC tools and parallel programming to involve more computational power to spilt the original problem into smaller sub-problems in order to

fit into memory [19]. Decomposition methods [12][6][16] are one of the highly invested methods in SVM. In the decomposition methods only a subset of variables is updated [16]. Based on this idea software as LIBSVM [4], SVM^{light} [10] have been implemented. Although software as LIBSVM and SVM^{light} using decomposition methods are efficient to solve SVM problems however when the problem size is large their performance is degraded since their computations are done in serial manner.

IV. Technologies for High Performance Computing

In this section, we briefly describe the technologies needed for the study of the PSVM communication and the improvement of the code.

A. Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is an architecture of parallel machines that use multiple processing elements to operate a single instruction on multiple data elements simultaneously. Today most compilers can automatically optimize simple code structures using vectors to take benefit from SIMD instructions. A common mistake is to make data dependent whereby the compiler can't optimize it [14]. In such cases the code needs to be restructured to make use of SIMD. The gain from SIMD is dependent on CPU architecture where the old SSE instructions gives a two fold improvement while the newest AVX with FMA can give up to 8 times improvement.

V. Preparation for Experiments

Before we describe the conducted experiments, we do a study of the SVM algorithm using the PSVM software and point out areas of interest that we have our main focus on. These parts are chosen based on their computational time relative to the total calculation time. The areas we focus on can benefit from HPC improvements which affect the total calculation time. Minor HPC improvements on heavy computational parts can have a large effect on the total computation time when the size of the problem is very large, and if the calculation time is not the issue of interest, we can improve accuracy for the same calculation time.

A. Used Factorizations

CF function calculates the Cholesky factorization of the original matrix A, i.e., CF calculates a lower triangular matrix G such that $A \approx GG^T$. CF solves a linear system by forward and backward substitutions. ICF approximates the original $n \times n$ matrix Q to a smaller $n \times p$ matrix H, i.e., $(Q - H^T H) \leq \epsilon$, where $p \ll n$ and ϵ is the error.

B. Algorithm

The solving process regarding (1) is done in two steps, first create,

$$Q \approx H^T H + \epsilon \tag{3}$$

then solve by IPM which is similar to solve by Newton steps [18]. The detailed information about Newton method used in IPM is mentioned by Boyd [3] and Mehrotra [15].

$$\triangle\boldsymbol{\lambda} = -\boldsymbol{\lambda} + \mathbf{vec}(\frac{1}{t(C-\alpha_i)}) + diag(\frac{\lambda_i}{(C-\alpha_i)})\triangle\mathbf{x} \tag{4}$$

$$\triangle\boldsymbol{\xi} = -\boldsymbol{\xi} + \mathbf{vec}(\frac{1}{t\alpha_i}) - diag(\frac{\xi_i}{\alpha_i})\triangle\mathbf{x} \tag{5}$$

$$\triangle\nu = \frac{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{z} + \mathbf{y}^T\boldsymbol{\alpha}}{\mathbf{y}^T\boldsymbol{\Sigma}^{-1}\mathbf{y}} \tag{6}$$

$$D = \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{7}$$

$$\triangle\mathbf{x} = \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \mathbf{y}\triangle\nu) \tag{8}$$

Minimize $P(\mathbf{w}, b, \boldsymbol{\xi})$ and $D(\boldsymbol{\alpha})$ along $\triangle\boldsymbol{\xi}$ and $\triangle\boldsymbol{\alpha}$ respectively (9)

$$\boldsymbol{\Sigma} = \mathbf{Q} + \mathbf{diag}(\frac{\xi_i}{\alpha_i} + \frac{\lambda_i}{C-\alpha_i}) \tag{10}$$

$$\mathbf{z} = -\mathbf{Q}\boldsymbol{\alpha} + \mathbf{1}_n - \nu\mathbf{y} + \frac{1}{t}\mathbf{vec}(\frac{1}{\alpha_i} - \frac{1}{C-\alpha_i}) \tag{11}$$

To compute $\Sigma^{-1}z$ the Sherman-Morrison Woodbury formula is used:

$$\begin{aligned}\Sigma^{-1}z &= (D+Q)^{-1}z \approx (D + HH^T)^{-1}z \\ &= D^{-1}z - D^{-1}H(I + H^T D^{-1}H)^{-1}H^T D^{-1}z \\ &= D^{-1}z - D^{-1}H(GG^T)^{-1}H^T D^{-1}z\end{aligned} \tag{12}$$

The equation above containing Σ^{-1} is the most interesting parts of the algorithm with respect to amount of computations and therefore it is divided up into sub steps as following,

$$E = I + H^T DH \tag{13}$$

$$GG^T = E \tag{14}$$

C. Complexity

By going through the SVM algorithm using the PSVM software, we calculate the complexity of both the computation and the communication. We denote the amount of rows on each CPU by η where η is calculated by the amount of training samples divided by the amount of cores, i.e., $\eta = \frac{n}{k}$. Notice that all equations are solved once per iteration except equation (3) which is only solved once. Equation (3) needs $O(p^3 + \eta p^2)$ computations and $O(log(k)(p+f))$ communication where f is the number of features, p is the amount of columns on each CPU, η is the amount of rows on each CPU, and k is the amount of cores. Equation (4), (5) and (7) are just vector operations and therefore has a complexity of $O(\eta)$ computations. Equation (6) is solved by using the result from (13) and (14). Since the system is solved by backward and forward substitution with GG^T for both Σ^{-1}z and Σ^{-1}y therefore we get a complexity of $O(p\eta + p^2)$ computation and $O(log(k)p)$ communication. In a similar manner we

calculate the corresponding complexities for equation (8), i.e., $O(p\eta + p^2)$ for computation and $O(log(k)p)$ for communication. The line search (9) has a complexity of $O(p^2)$ computations. The equation (13) is a matrix multiplication and therefore has the complexity $O(\eta p^2)$ for computations and $O(log(k)p^2)$ for communication. The last equation (14) which is the CF is done serially on the master and has complexity $O(p^3)$.

Among the above mentioned equations, i.e., equations (3) to (14), we focus mainly on equations (13) and (14), since they are the most computationally expensive functions relative to the total computation time and thus they are more interesting for further study and investigation for potential improvements.

VI. Experimental Design

Our goal is to explore, and study the behaviour of the SVM algorithm regarding high performance computing point of view. In this respect, we choose an exploratory approach to discover new insights regarding SVM algorithm that can affect the class-prediction accuracy and the training time using PSVM. PSVM is implemented by Chang et al.[20]. We improve the PSVM code regarding structure, memory allocation, de-allocation and parallelism point of view as mentioned in section IV-A. We conducted experiments 1) to study the impact of changing hyperparameter C and γ on total training time by considering target class-prediction accuracy, 2) to study the impact of changing the number of columns p on the training time and the class-prediction accuracy using Gaussian and Laplacian kernels and to study the trade-off between the class-prediction accuracy and the time efficiency by changing p settings, 3) to evaluate the existence of a threshold for the appropriate number of computational nodes in order to get the advantage of parallelism as much as possible.

We measure the class-prediction accuracy of the models based on the number of correct predictions among all the correct and incorrect predictions.

$$Accuracy = \frac{T_- + T_+}{T_- + T_+ + F_- + F_+}$$

Where T_+, T_-, F_+ and F_- are true positive, true negative, false positive and false negative respectively.

The reduced number of columns in ICF is denoted by $p = n^r$ where n is the number of samples and r is the reducing ratio between 0 and 1. For all the experiments unless stated otherwise, we use the improved PSVM and the publically available *cod-rna, covertype, webspam* and *url* datasets provided by UCI data repository for machine learning [11] and by Fan et al. [5].

A. Experiment 1: Sensitivity of PSVM Regarding C and γ

One of the challenges of SVM problems using Gaussian and Laplacian kernels is to choose appropriate values for hyperarameter C and γ [8][7], since the class-prediction accuracy and time efficiency [13] are influenced by these parameters. Intuitively, γ means how far the influence of a single training sample is. A large value of γ shows the low influence of a single training sample while a low value of γ shows the high influence. The γ parameter has an inverse relationship with the radius of influence of support vectors [2]. The hyperparameter C makes a balance between the misclassification and the maximum margin. A low value of C makes the decision surface smooth while a large value of C gives freedom to the model to choose more support vectors among samples that results in more precise and accurate classification of the training samples [2]. One of the common way to find a suitable hyperparameter C and γ is cross-validation [9], however finding the best value of these parameters are still unclear.

In the first experiment, we study the impact of C and γ parameters on the sensitivity of training time meaning how much the training time varies by changing the C and γ parameters. In this experiment, we chose C between 0.1, 1, ... , 100000 and chose γ between 0.1, 1, ... , 1000. We conduct this experiment on two datasets, *cod-rna* and *covertype* and we study the total calculation time for training the samples.

B. Experiment 2: Reduced Column p

The reduced number of columns p in ICF has impact on the class-prediction accuracy and the training time and finding an appropriate value for p in ICF is non-trivial and controversial. Larger value of p results in higher class-prediction accuracy but slower total training time while smaller value of p results in fast training of samples but poor class-prediction accuracy. Although Chang et al. [20] suggest $p = \sqrt{n}$ based on their experimental results, however the trade-off between the class prediction accuracy and the time efficiency by changing p has been unclear. It is also unclear how the value of p influences different kernels.

In experiment 2, we study the impact of different p settings on the class-prediction accuracy and the training time. To study the performance of PSVM for different p settings, we divide the second experiment into sub-experiments. In the first sub-experiment, we have chosen a range of different p from $n^{0.3}$ to $n^{0.6}$ for the fixed values of C and γ and we measure the class-prediction accuracy. In the second sub-experiment, we have improved the first sub-experiment by choosing the best C and γ parameters for each p settings and we measure the class-prediction accuracy. In the third sub-experiment, we study the impact of p settings on total training time and training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts. In addition, we compare the training time regarding the original PSVM with the improved PSVM and do a short study of how the changes that we did affect the proportions. We conduct experiment 2 for webspam dataset with 300000 samples and 254 features and covertype datasets with 500000 samples and 54 features. We use both Gaussian and Laplacian kernel functions in this experiment.

C. Experiment 3

Although using HPC tools is promising for higher performance, however due to communication overheads choosing

appropriate number of computational powers such as number of computational nodes are still non-trivial. Based on the Amdahl's law , the maximum speed up of a program using parallel computing with multiple processor is limited regardless of the number of processors.

In experiment 3, we study the relation between the complexities we found in earlier chapter and the actual values when running the datasets. We run the experiment on improved PSVM with 8, 16, 32, 64, 128, and 256 computational nodes and we measure the total training time and the training time regarding E, CF, updating variable. For clarity, we measure the proportional training time on heavy computational parts of SVM algorithm as calculation of E, CF, ICF, Updating variables and other parts and we also measure the time for communication as the communication in E and the communication for Updating variables.

VII. Results

In this section we present the results of all three experiments and the corresponding sub-experiments.

A. Experiment 1

The results of first experiment are presented in Tables I and II. Table I represents the total time for training 59535 samples with 8 features for cod-rna dataset. In the first experiment, we choose a target class-prediction accuracy inside 10 percentage point of the best accuracy obtained. The red cells in table I shows that for the given C and γ the target accuracy is not achieved. Table I shows no trends between different settings of C and γ and the total training time. As the table shows the lowest total training time is 8 times slower than the highest total training time. The results of the first experiment on covertype dataset is shown in table II and it represents the total training time for 500000 samples with 54 features. In table II the target accuracy for $\gamma = 0.1$, $\gamma = 1$ and $\gamma = 10$ is not achieved. As the table shows the lowest total training time is 5.5 times slower than the highest total training time and same as table I, we do not detect any trends between C and γ and total training time. The kernel function that is used during this experiment is Gaussian kernel.

TABLE I
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COD-RNA DATASET WITH 59535 SAMPLES AND 8 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	3.33	1.83	1.28	1.25	0.73
C=1	1.87	1.23	1.06	0.79	0.68
C=10	9.2	1.09	1.07	0.77	0.7
C=100	9.24	1.35	1.59	1.13	1.04
C=1000	8.91	9.11	8.69	0.99	9.02
C=10000	8.99	8.95	9.18	1.27	8.98
C=100000	9.17	8.96	9.24	1.96	10.41

B. Experiment 2

Figure 1 is related to the first sub-experiment in experiment 2 and it shows the class-prediction accuracy with respect to different column sizes. We have selected C and γ parameters as $C = 64$ and $\gamma = 2$ mentioned by Hsieh, Si and Dhillon [8] for webspam dataset. As figure 1 shows by increasing the number of columns, p from $n^{0.3}$ to $n^{0.5}$, the class-prediction accuracy degrades drastically, where for $p = \sqrt{n}$ the accuracy reaches it's lowest value, by increasing the value of p more than $n^{0.5}$, the class-prediction accuracy increases. We observe that the class-prediction accuracy is unstable by increasing the number of columns and for the fixed values of C and γ for all p columns.

TABLE II
NUMERICAL RESULTS OF TOTAL TRAINING TIME WITH RESPECT TO DIFFERENT C AND γ SETTINGS FOR COVERTYPE SCALED DATASET WITH 500000 SAMPLES AND 54 FEATURES USING GAUSSIAN KERNEL

Parameter	γ=0.1	γ=1	γ=10	γ=100	γ=1000
C=0.1	332.45	300.26	225.48	241.24	133.39
C=1	85.25	96.55	90.96	137.25	95.99
C=10	42.81	55.24	77.88	104.35	106.94
C=100	52.27	63.12	77.5	71.31	135.23
C=1000	68.58	80.54	92.37	42.99	178.69
C=10000	88.54	137.79	171.65	141.97	415.62
C=100000	284.75	178.96	367.56	417.05	414.29

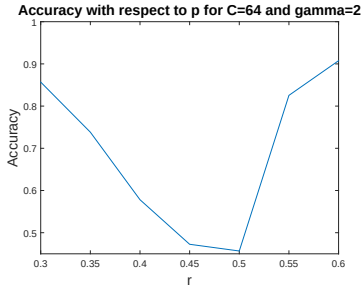

Fig. 1. The class-prediction accuracy with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using C = 64, $\gamma = 2$ and Gaussian kernel function.

Figure 2 shows the second sub-experiment results for both Gaussian and Laplacian kernel functions. This figure gives a better insight that we can observe that replacing C and γ parameters with the best C and γ for each r results in stable class-prediction accuracy where $r = log(p)/log(n)$.

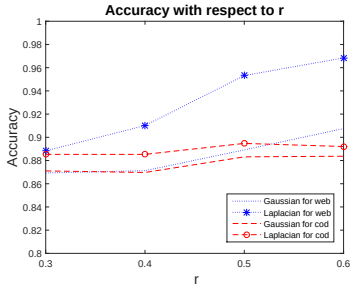

Fig. 2. The class-prediction accuracy with respect to different r ($r = log(p)/log(n)$) for webspam and cod-rna datasets using best C and γ for each r.

Figure 3 shows that the best C and γ stays close when p is changed. Figure 4 is related to the third sub-experiment of experiment 2 and shows the elapsed training time for E, CF, updating variables and other part of SVM algorithm regarding different p settings in webspam dataset. The upper sub-plot shows the elapsed time in seconds and the lower sub-plot shows the proportion time of each parts. The increased proportion of CF and E follows the results of the complexity analysis earlier. The difference between original and the improved PSVM can be seen in figure 5. As figure 5 shows the elapsed time regarding parts E and CF decreases in improved PSVM compared to the original PSVM when r increases.

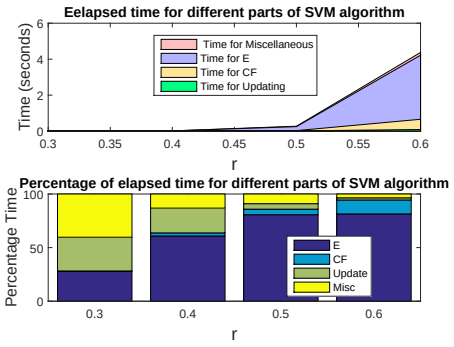

Fig. 4. The elapsed time for different parts of SVM algorithm with respect to different column numbers (p) for webspam dataset with 300000 samples and 254 features using Laplacian kernel function and best C and γ for each p.

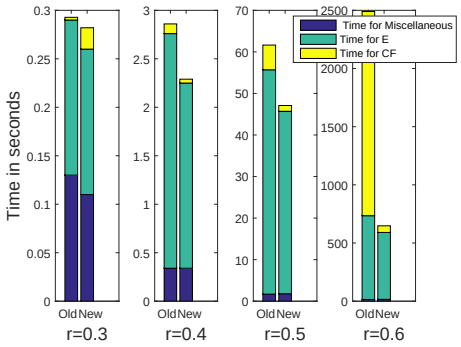

Fig. 5. The comparison between the original PSVM (Old) and the improved PSVM (New) software with respect to r.

C. Experiment 3

Figure 6 shows the training time and the time for calculating E, CF, updating variables of the SVM algorithm with respect to the number of computational nodes for covertype dataset. The upper sub-plot shows how the corresponding training time decreases by increasing the number of nodes from 8 to 64 while increasing the number of nodes more than 64 nodes does not show further improvement in training time. The lower sub-plot gives a better insight about the proportion of training time in E, CF, updating variables, ICF, and other parts of PSVM algorithm for covertype dataset along with communication time for E and communication time for updating variables. With this dataset, 128 nodes were enough to reach the threshold predicted in the complexity analysis.

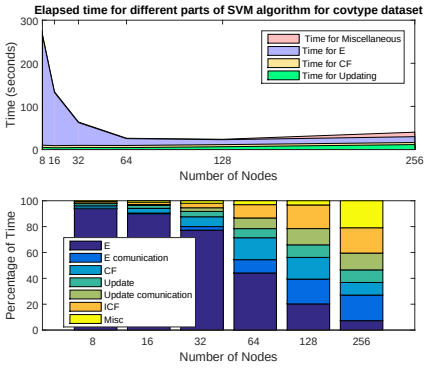

Fig. 6. The elapsed time for different parts of SVM algorithm with respect to the number of nodes, N for covtype dataset with 500000 samples and 54 features. Note the large proportion of communication at 128 nodes.

Figure 7 shows the same experiment as above for URL dataset. The upper sub-plot in figure 7 shows how the corresponding training time decreases by increasing the number of nodes from 8 to 128 while increasing the number of nodes more than 128 shows not remarkable improvement in the training time. The lower sub-plot gives a better insight about the proportion of training time in E, communication in E, CF, update variables, communications for updating variables, ICF, and other parts of the PSVM algorithm for URL dataset.

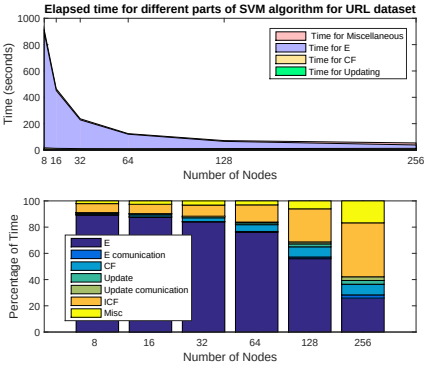

Fig. 7. The elapsed time for different parts of SVM algorithm with respect to the number of nodes N for URL dataset with 2150000 samples and 3231961 features.

VIII. Conclusion

In experiment 1, we study the impact of C and γ parameters on the training time considering the target accuracy. We did not find any interesting trend in the training time by changing these parameters. However this experiment helped us to get better insight about experiment 2 where we observed an improvement in the class-prediction accuracy while

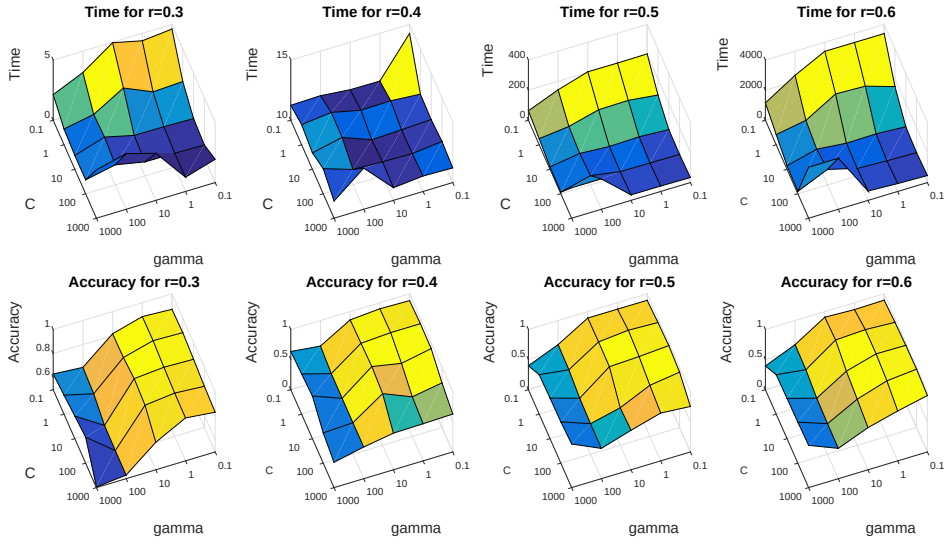

Fig. 3. The training time and accuracy with respect to different column numbers (p) for cod-rna considering different C and γ settings for Laplacian kernel.

increasing the number of columns. The result of experiment 2 showed that choosing appropriate value of p affects the choice of C and γ, this is clearly shown by figure 1. The common way to choose C and γ is done by cross-validation. The result of experiment 2 showed that choosing the best values for C and γ for special column size p is not necessarily the best value for another p. The complexity analysis for CF did predict a fast growth of calculation time for CF when p is increased which could clearly be seen by experiment 2. In figure 5, the original software was tested against the improved software for different p and showed 4 times improvement of performance by our modification on PSVM at large p because of the CF calculation. Already at smaller p we got an 20% improvement on the calculation of E. In experiment 3, we showed the existence of a threshold between the training time and the number of cores as predicted in a complexity analysis.

IX. Acknowledgements

Our experiments were using the Triolith system from National Supercomputer Center (NSC) at Linkoping University.

References

[1] A fast parallel optimization for training support vector machine. In Petra Perner and Azriel Rosenfeld, editors, *Machine Learning and Data Mining in Pattern Recognition*, volume 2734 of *Lecture Notes in Computer Science*. 2003.

[2] scikit-learn developers (BSD License) 2010 2014. RBF SVM parameters. http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, 2015.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[5] Chih-Chung Chang, Chih-Jen Lin, and Rong-En Fan. LIBSVM data: Classification, regression, and multi-label. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, 2015.

[6] Fu Chang, Chien-Yang Guo, Xiao-Rong Lin, and Chi-Jen Lu. Tree decomposition for large-scale SVM problems. *The Journal of Machine Learning Research*, 11:2935–2972, 2010.

[7] Asdrúbal López Chau, Xiaoou Li, and Wen Yu. Support vector machine classification for large datasets using decision tree and fisher linear discriminant. *Future Generation Computer Systems*, 36:57–65, 2014.

[8] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. A divide-and-conquer solver for kernel support vector machines. *arXiv preprint arXiv:1311.0914*, 2013.

[9] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification, 2003.

[10] T. Joachims. Making large-scale SVM learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[11] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[12] Chih-Jen Lin. On the convergence of the decomposition method for support vector machines. *Neural Networks, IEEE Transactions on*, 12(6):1288–1298, 2001.

[13] Gaëlle Loosli and Stéphane Canu. Comments on the "core vector machines: Fast SVM training on very large data sets". *J. Mach. Learn. Res.*, 8:291–301, May 2007.

[14] S. Maleki, Yaoqing Gao, M.J. Garzaran, T. Wong, and D.A. Padua. An evaluation of vectorizing compilers. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 372–382, Oct 2011.

[15] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[16] John Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methodssupport vector learning*, 3, 1999.

[17] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson International Edition. Pearson Addison Wesley, 2006.

[18] Tamás Terlaky. *Interior point methods of mathematical programming*, volume 5. Springer Science & Business Media, 1996.

[19] Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *The Journal of Machine Learning Research*, 7:1467–1492, 2006.

[20] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. Parallelizing support vector machines on distributed computers. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. Curran Associates, Inc., 2008.