Parallel SVM Using OpenMp and MPI

Introduction

The statistical analysis in any field involves processing large datasets which have grown in size exponentially over the years and will continue to grow so. As the data generated by multiple sources explode, not only the space complexity increases but, the computation time over such large training datasets also increases. Thus, one of the biggest challenges faced while applying any Machine Learning algorithm to the real world datasets is to be able to process the vast amount of data in a reasonable amount of time.

One of the most popular approach for ‘off-the-shelf’ supervised learning is support vector machines. The training stage for Support Vector Machines i.e. building the classifier using training set, involves a dense convex quadratic optimization problem (QP). Solving this optimization problem is computationally expensive due to Hessian Metrix. The standard SVM training has O(m3) time and O(m2) space complexities, where m is the training set size. Thus scaling QP for large values of m (real world datasets may have size ranging equal to or even more than 10000000 examples in about 20-30 dimensions) is an important issue for making the problem tractable. The wall clock time for building the classifier using linear SVM learning algorithm which runs sequentially becomes infeasible for general purpose solver. Also the parameters required for the training procedure, i.e. the Gamma and Cost values are not always known. The most common and naive approach used to fix the parameter is Grid Search. Where we cross validate the training set for different pair of parameters and then train the model with parameters giving highest cross-validation accuracy.

In this project I worked with a dataset of about 10000 training examples having 500 features. This is a moderately large dataset with high dimension feature space. I aim to find out how much efficiency in terms of time is achieved by using hybrid approach to parallelize over serial implementation.

Approach

In this project I have parallelized training procedure for SVM algorithm using Hybrid approach i.e. OpenMp and MPI. OpenMP works effectively on shared memory systems, while MPI can exploit the computation power of multiple nodes using message passing, it has a significant communication overhead.

Here I have divided the problem space in two layers of parallelization. First I implemented the Grid Search - to find the best pair of Gamma and Cost value and divide the problem among various MPI threads. Since each Gamma-Cost pair is independent of the other, there is no communication required in between. Each MPI thread can find the accuracy for the set of pairs assigned to it and then inform the master (root) thread, which in turn selects the global best pair and train then output the final train model based on best fit.

The second layer of parallelization divides the training set examples among various OpenMp threads i.e. each MPI rank further delegates the task to 'd' OpenMp threads. For each training set example we need to solve the Quadratic optimization problem per iteration. For a moderately sized dataset, in case of MPI(like the one I used for this project) this would add more communication overhead than the speedup that could be achieved.

I parallelize the Grid Search during the training procedure using MPI. There are 16 different values of Gamma (8, 4, 2, …… 0.0625, ….. 0.000061035) and 16 different values of Cost (0.03125, … 2^13, 2^14). Thus we have to find the best pair from the set of 256. Each MPI node cross validate for all the Gamma values paired with 16/n different Cost values.

For e.g.. If there are 8 MPI node and 4 OpenMp processes, then each node check for 2 values of Cost. In that case MPI rank 7 test for

cost 2^13 x gamma (8, 4, 2, …… 0.0625, …..0.000061035) and

cost 2^14 x gamma (8, 4, 2, …… 0.0625, …..0.000061035)

Also, it further divides the data into 4 parts. Thus OpenMp thread 1 of rank 7 will test all the above mentioned cost and gamma pairs for (dataset size)/4 examples, starting of from 0. Thread 2 will get next chunk and so on.

Each thread sends their locally solved QP to the MPI rank, which then tests for next pair and after completing the accuracy for all pairs send the final accuracies to the root thread (i.e. thread with mpi rank 0). Root thread then selects the pair with maximum accuracy and trains the model in parallel using ‘d’ OpenMp threads and stores it to the model file.

Here I have compared performance on the basis of computation time and speedup achieved against single node for different number of nodes (2,4,8,16) and different number of cores (1,2,4,8,16,32) per node. Against node = 1 with single core i.e. full sequential code.

Code snippet of libsvm library parallelized using OpenMp

Results

It is interesting to note that while the time to solution decreases as we increase the number of nodes, it infect starts to increase after number of

cores per node are increased more than 8. This is because in case of strong scaling i.e. when size of problem/dataset is fixed, after a certain point further dividing the problem adds more management overhead than the computational speedup achieved.

Below are the Graphs showing time in seconds to train the model for n = (2,4,8,16) nodes with d = (1,2,4,8,16,32) cores per node. It shows how the performance changes as we increase number of cores per node.

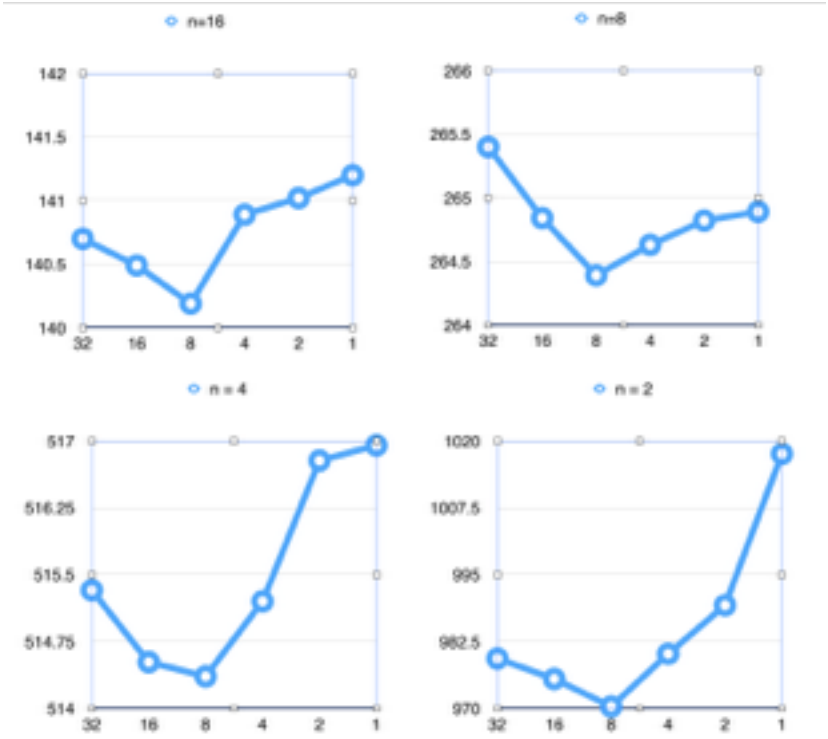

Figure 1 Y- axis shows time and X-axis shows number of cores/OpenMp Thread per node for MPI rank n = 16, 8 , 4 , 2 respectively

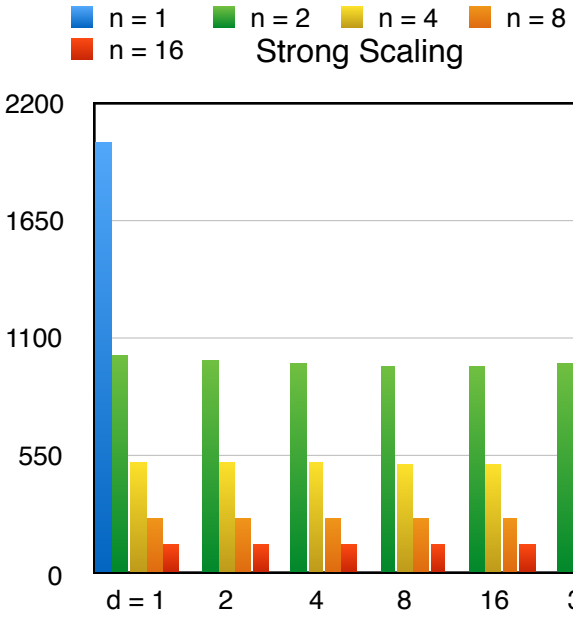

Figure 2 Here n is the number of MPI ranks and d is the number of OpenMp thread per rank.

The effect of increasing number of MPI nodes to parallelize grid search can be clearly seen in table below.

Number of MPI Ranks	MPI Ranks				
		2	4	8	16
Number of OpenMP threads	1	1017.59	516.95	264.89	141.2
	2	989.28	516.78	264.82	141.02
	4	980.20	515.20	264.63	140.89
	8	970.35	514.35	264.39	140.19
	16	975.32	514.51	264.84	140.49
	32	979.32	515.32	265.4	140.7

Figure 3 Exact wall clock time for various cases

Time taken by sequential code = 2018.64s i.e n = 1, d = 1 (n : #MPI ranks, d : #OpenMP threads)

Below is the heat map showing Accuracies for various pairs of Gamma and Cost and it can be observed that wall clock time of the problem can be further reduced by optimized grid search algorithm, but since the aim of this project was exploit parallelism, I have not implemented that.

Figure 4 Heatmap for accuracy. Y-axis Cost, X-axis Gamma

Tools / Library

For plotting the heat map of accuracies achieved with different parameter set, I used gnuplot. For parallel training I have used OpenMP and MPI library and libsvm library is used for the SVM algorithm.