10.48047/jocaaa.2024.33.08.132

Efficient Parallel Optimization of Multiclass SVM using 1-Factorization Approach

Vijayakumar H. Bhajantri[1], Shashikumar G. Totad[2], Geeta R. Bharamagoudar[3]

[1,2]School of Computer Science and Engineering, KLE Technological University, Hubballi, Karnataka, India.
[3]Department of Computer Science and Engineering, KLE Institute of Technology, Hubballi, Karnataka, India.

Abstract

Support Vector Machines (SVMs) are widely used for multiclass classification tasks due to their strong generalization capabilities. However, their computational complexity increases significantly with large datasets, making parallel optimization essential. This paper presents an efficient parallel optimization approach for multiclass SVMs using 1-Factorization in a distributed computing environment. The 1-Factorization technique efficiently decomposes the classification problem into independent binary SVM sub-problems, enabling parallel computation across multiple processing nodes. By leveraging distributed computing frameworks, we reduce training time while maintaining classification accuracy. Experimental results demonstrate significant speedup and scalability improvements compared to traditional optimization techniques. Our approach enhances the feasibility of SVMs for large-scale applications, making them more suitable for real-world scenarios requiring high-performance classification.

Keywords: Machine Learning, Big Data, ADMM, Symmetric ADMM, Support Vector Machine, Classification, Parallel and distributed computing, Crammer-Singer, Weston-Watkins, Multiclass.

1. Introduction

In the era of big data, machine learning has emerged as a fundamental approach for handling complex classification problems. Among various classification models, Support Vector Machines (SVMs) have proven to be highly effective due to their strong theoretical foundation and ability to generalize well to unseen data. However, traditional SVMs are inherently binary classifiers, making them less efficient for multiclass classification tasks. Several strategies have been proposed to extend SVMs to multiclass problems, including one-vs-one (OvO)**,** one-vs-all (OvA)**,** and direct multiclass formulations. One such notable approach is the Weston-Watkins multiclass SVM algorithm, which directly optimizes a single objective function for multiple classes, reducing the number of classifiers required compared to OvO and OvA approaches.

Despite the advantages of multiclass SVMs, their computational complexity increases significantly with large datasets. Training an SVM requires solving a quadratic optimization problem, which becomes infeasible when dealing with high-dimensional data or large-scale applications. To address this issue, parallel optimization techniques have been proposed to accelerate SVM training by distributing computations across multiple processors or nodes. This paper explores an effective parallel optimization approach for multiclass SVMs using the 1-

10.48047/jocaaa.2024.33.08.132

Factorization method in a distributed computing environment, significantly enhancing computational efficiency while maintaining classification accuracy.

The demand for fast and scalable machine learning algorithms has grown exponentially with the advent of large datasets in various domains such as healthcare, finance, image processing, and natural language processing. SVMs, originally introduced by Vladimir Vapnik, have remained one of the most widely used classification techniques due to their ability to find an optimal hyperplane that maximizes the margin between classes. However, when extended to multiclass classification, standard SVM formulations become computationally expensive.

The Weston-Watkins multiclass SVM algorithm provides an elegant solution by formulating the classification problem as a single optimization function rather than decomposing it into multiple binary classification tasks. Unlike the OvO and OvA approaches, which train multiple binary SVMs separately, Weston-Watkins SVM optimizes a single objective function that simultaneously learns decision boundaries for all classes. This formulation ensures better computational efficiency and avoids inconsistencies that may arise from independent binary classifiers.

Despite these advantages, training a Weston-Watkins multiclass SVM on large datasets still poses a significant computational burden. The optimization problem involves solving a large quadratic programming (QP) formulation, which is computationally intensive and often requires long training times. This limitation motivates the need for parallel and distributed computing strategies to improve efficiency.

Parallel computing techniques have been widely adopted in machine learning to speed up training processes and handle large-scale data efficiently. Parallelization can be achieved at different levels, such as:

1. Data Parallelism – Splitting the dataset into smaller subsets and distributing them across multiple processors or nodes.
2. Model Parallelism – Dividing the model parameters and computing them simultaneously on different computing units.
3. Hybrid Parallelism – Combining both data and model parallelism for optimal efficiency.

For multiclass SVMs, a straightforward parallelization approach involves training binary classifiers independently in one-vs-one or one-vs-all settings. However, in the Weston-Watkins SVM formulation, the entire optimization problem is solved in a single step, making it less straightforward to parallelize. Our proposed approach leverages 1-Factorization to decompose the problem into independent sub-problems that can be efficiently parallelized across multiple computing nodes.

1-Factorization is a mathematical technique used in graph theory to decompose a complete graph into disjoint perfect matchings. In the context of multiclass SVMs, 1-Factorization can be applied to break down the multiclass problem into smaller independent tasks, each of which can be optimized separately.

Why 1-Factorization?

10.48047/jocaaa.2024.33.08.132

- Reduces Computational Complexity: Instead of solving a massive optimization problem, 1-Factorization partitions the problem into smaller independent sub-problems, making them computationally feasible.
- Enhances Parallelism: Since the sub-problems are independent, they can be executed simultaneously across multiple processors in a distributed environment.
- Preserves Classification Accuracy: The decomposition ensures that the classification boundaries remain consistent across all classes, avoiding inconsistencies seen in traditional binary decomposition methods.

2. Literature Review

This literature survey provides a comprehensive review of multiclass Support Vector Machine algorithms, distributed computing approaches, focusing on distributed optimization algorithms, their classifications, implementation techniques, and applications.

2.1 Introduction to Multiclass Support Vector Machines (SVMs)

Support Vector Machines (SVMs) have been widely used in machine learning and pattern recognition due to their strong theoretical foundation and ability to handle high-dimensional data efficiently. However, traditional SVMs are inherently binary classifiers, meaning they are designed to separate two distinct classes [1]. Extending SVMs to multiclass classification has been a long-standing research challenge, and various techniques have been proposed to address this limitation

2.1.1 Traditional Approaches to Multiclass SVMs

Several methods have been proposed to extend SVMs to multiclass classification, with the most common being:

- One-vs-One (OvO): In this method, $\frac{K(K-1)}{2}$ binary classifiers are trained for a $K-class$ problem, and classification is performed using a voting mechanism. The class with the highest votes is selected as the final output. While effective, this approach becomes computationally expensive for large K.
- One-vs-All (OvA): This technique trains K binary classifiers, where each classifier distinguishes one class from all others. The class with the highest confidence score is selected. While simpler than OvO, it suffers from potential class imbalance issues [2].
- Directed Acyclic Graph SVM (DAG-SVM): This method organizes binary classifiers in a directed graph structure, reducing the number of comparisons needed for classification. Although it speeds up classification, training complexity remains high.
- Error-Correcting Output Codes (ECOC): This technique represents multiclass problems using a binary coding matrix, allowing binary classifiers to collectively determine the final class.

10.48047/jocaaa.2024.33.08.132

These approaches decompose the multiclass problem into multiple binary sub-problems, leading to increased computational overhead. To address this limitation, direct multiclass SVM formulations have been introduced.

2.2 Direct Multiclass SVM Formulation

Instead of decomposing the problem into multiple binary classifiers, direct multiclass SVM formulations aim to solve a single optimization problem for all classes.

2.2.1 Weston-Watkins Multiclass SVM

One of the most well-known direct multiclass SVM formulations was introduced by Weston and Watkins (1999) [3]. Their approach optimizes a single objective function, ensuring that the decision boundaries for all classes are learned simultaneously. Unlike OvO and OvA methods, which train separate binary classifiers, the Weston-Watkins SVM optimizes all class decision boundaries within a single optimization framework, reducing inconsistencies in classification.

Mathematically, the Weston-Watkins formulation minimizes the following objective function:

$$\sum_{i=1}^{n}\sum_{j\neq y_i} \max\left(0,1+\ w_j^T x_i - w_{y_i}^T x_i\right)$$

Where w_j and w_{y_i} represent weight vectors for class j and the correct class y_i respectively. This approach ensures that the correct class decision function is greater than all others by at least a margin of 1**.**

2.2.2 Crammer-Singer Multiclass SVM

Crammer and Singer (2001**)** further improved direct multiclass SVMs by jointly optimizing all class decision boundaries**,** making it more efficient than Weston-Watkins SVM [4]. However, the training complexity of these direct formulations is significantly higher than decomposition-based methods**,** making parallel computing essential**.**

The objective function of Crammer-Singer multiclass SVM formulation is:

$$\min_{w}\frac{1}{2}\sum_{k=1}^{K}\|W_k\|^2 + C\ \sum_{i=1}^{n}\max_{j\neq y_i}\left(1+W_j^T x_i - W_{y_i}^T x_i\right)$$

With reformulated constraints

$$\min_{w}\frac{1}{2}\|W\|_F^2 + C\sum_{i=1}^{n}\xi_i$$

subject to

$$w_{y_i}^T x_i - \ w_k^T x_i\ \geq 1 - \ \xi_i, \forall k \neq y_i, \xi_i \geq 0, i = 1,\ldots,n$$

10.48047/jocaaa.2024.33.08.132

Here: $W \in R^{Kxd}$: $A\ matrix\ where\ each\ row\ w_k is\ the\ weight\ vector\ for\ class\ k, \xi_i\ is$ the slack variables for misclassification.$C > 0$ is the regularization parameter. $W_F^2\ is\ the$ Frobenius norm of the regularization.

2.3 Parallel Optimization for Multiclass SVMs

Support Vector Machines (SVMs) have been widely used in classification tasks due to their ability to generalize well on unseen data. However, training SVMs, particularly for multiclass classification, poses computational challenges due to the high dimensionality of data and the quadratic programming nature of the optimization problem. To address these issues, parallel optimization techniques have been explored to enhance the efficiency and scalability [5] of SVM training. Multiclass classification using SVMs can be approached in multiple ways, such as One-vs-All (OvA), One-vs-One (OvO), and direct multiclass SVM formulations. Each of these methods involves solving multiple binary classification problems or a single large-scale optimization problem, both of which can be computationally expensive. Traditional optimization methods, such as Sequential Minimal Optimization (SMO) and Quadratic Programming (QP), are often inadequate for handling large datasets efficiently, thus necessitating parallel optimization techniques [6][7].

Given the computational challenges associated with multiclass SVMs, researchers have explored various parallelization techniques to improve efficiency. Parallel optimization strategies can be categorized based on how they distribute computation across multiple processors or computing nodes. Below are the key techniques explored in the literature:

2.3.1 Data Parallelism

Data parallelism involves splitting the training dataset into smaller subsets and distributing them across multiple processors. Each processor independently trains an SVM on its assigned subset, and the results are then aggregated to form the final model. This method is particularly effective in distributed computing environments, such as cloud-based infrastructures.

- Hsu and Lin (2002) [24] implemented a parallelized SMO algorithm that divides data among multiple processors and iteratively updates the model using a shared-memory approach.
- Graf et al. (2005) [25] proposed a parallelized kernel SVM where subsets of data are trained separately before being combined, reducing training time significantly.

2.3.2 Task Parallelism

Task parallelism assigns different computational tasks to different processors. This approach is beneficial for training multiclass SVMs, where individual binary classifiers in OvA or OvO strategies can be trained concurrently.

- Keerthi et al. (2008) [6] demonstrated an efficient parallel training mechanism by distributing binary classifiers across multiple cores, significantly reducing computation time.

10.48047/jocaaa.2024.33.08.132

- Joachims (1999) introduced an optimization framework that leverages multiple cores to perform hyperplane updates in parallel, improving convergence speed.
- Implementation Details: Task parallelism can be implemented using thread-based parallelism where different classifiers or optimization steps are assigned to independent threads or processors. It is often synchronized using shared memory or message-passing techniques to ensure consistency across computations.
- Advantages: Task parallelism is particularly useful for large-scale multiclass problems where multiple independent tasks, such as training different binary SVM classifiers, can be executed simultaneously.
- Challenges: The synchronization and communication overhead between tasks can impact performance, especially in cases where tasks have dependencies or require frequent model updates [8] [9].

2.3.3. Model Parallelism

Model parallelism distributes the model parameters across different processors instead of the data. This method is particularly useful for extremely large SVMs where storing and updating all parameters on a single machine is impractical.

- Zhu et al. (2009**)** developed a parallelized decomposition method that splits the weight vector among multiple processors, allowing faster training.
- Chang et al. (2010) proposed a distributed QP solver for large-scale SVMs, achieving improved efficiency and scalability.
- Implementation Details: In model parallelism, different parts of the SVM model (such as support vectors, weight vectors, or kernel matrices) are distributed across multiple computing nodes. Each node updates only its assigned parameters, and periodic synchronization is performed to ensure model consistency.
- Advantages: This approach is highly effective for handling large datasets with high-dimensional feature spaces, reducing memory constraints on a single processor.
- Challenges: Synchronization of distributed parameters can be complex, and communication overhead may slow down convergence in distributed environments [10] [11].

2.3.4. Hybrid Parallelism

Hybrid parallelism combines data and task parallelism to maximize computational efficiency. This approach is often implemented in high-performance computing environments where both data and computational resources are distributed.

- Catanzaro et al. (2008) introduced GPU-accelerated SVMs that leverage both parallel computation for kernel evaluations and concurrent binary classifier training.
- You et al. (2016) developed a distributed framework that integrates both data and task parallelism, achieving significant speedup over traditional training methods [12] [13].
- Implementation Details: Hybrid parallelism is typically employed in heterogeneous computing environments where multiple GPUs, CPUs, and distributed clusters are used together. Data parallelism ensures that training data is processed in parallel, while task

10.48047/jocaaa.2024.33.08.132

parallelism ensures that multiple computations, such as kernel evaluations and weight updates, are performed concurrently.

- Advantages: Hybrid parallelism optimally utilizes available computing resources, providing better scalability and faster convergence.
- Challenges: Managing resource allocation, communication synchronization, and load balancing across different parallelization strategies can be complex [14].

2.4 Parallelization Frameworks and Libraries

Several frameworks and libraries have been developed to facilitate parallel SVM training: CUDA and OpenCL: GPU-based parallelization of SVMs for large-scale datasets. Apache Spark and Hadoop: Distributed computing frameworks that enable large-scale parallel training of SVMs. MPI (Message Passing Interface)**:** A high-performance computing approach for implementing distributed SVM training across multiple nodes.

2.5 Performance Comparison of Parallel Techniques

Performance metrics such as speedup, scalability, and classification accuracy have been widely studied to compare different parallel optimization techniques. Key findings from the literature include: Speedup and Scalability**:** GPU-based implementations achieve speedups of up to 50x compared to traditional CPU-based training. Accuracy Trade-offs**:** While parallel training reduces computational time, careful hyperparameter tuning is required to maintain classification accuracy [15]. Memory Efficiency**:** Distributed SVM frameworks, such as those using Apache Spark, demonstrate better memory efficiency for large datasets.

2.6 Distributed Computing Approach

Distributed computing is a paradigm that enables the execution of computational tasks across multiple interconnected computing nodes to enhance efficiency, scalability, and fault tolerance. With the rapid increase in data volumes and complexity, distributed optimization algorithms have gained significant importance in solving large-scale problems across various domains, including machine learning, numerical analysis, and operations research.

2.6.1 Distributed Computing and Optimization

Distributed computing involves dividing a computational task into subtasks and distributing them among multiple nodes, which collaborate to achieve a common goal. The efficiency of this approach depends on network communication, synchronization mechanisms, and task scheduling. Distributed optimization algorithms extend this concept by enabling optimization problems to be solved across multiple computing nodes while ensuring convergence and optimality. These algorithms are essential in scenarios where centralized processing is infeasible due to memory constraints, high computational costs, or privacy concerns [16] [17].

Distributed optimization algorithms can be broadly classified into the following categories:

10.48047/jocaaa.2024.33.08.132

1. Gradient-Based Methods

Gradient-based optimization techniques are widely used for solving convex and non-convex problems in distributed settings. Some of the key methods include:

- Distributed Gradient Descent (DGD)**:** Each node computes the local gradient and shares it with neighboring nodes to update the model parameters iteratively.
- Dual Averaging Methods**:** These methods accumulate gradient information over iterations, improving convergence rates.
- Alternating Direction Method of Multipliers (ADMM)**:** A powerful technique that decomposes optimization problems into smaller sub-problems, which are solved independently before combining results.

2. Consensus-Based Optimization

Consensus-based methods ensure that distributed nodes reach a common solution by iteratively exchanging information. These methods are particularly useful in decentralized networks where direct coordination is limited [18].

- Consensus ADMM: Extends ADMM by incorporating consensus constraints to ensure convergence across multiple nodes.
- Distributed Newton's Method: Uses second-order information to achieve faster convergence while maintaining consensus constraints.
- Multi-Agent Optimization: Used in sensor networks and distributed control systems, where agents collaboratively optimize local objectives while maintaining global consistency.

3. Primal-Dual Methods

Primal-dual methods solve optimization problems by iteratively updating primal (decision) and dual (constraint) variables. These methods are useful for handling constrained optimization problems in distributed environments.

- Dual Decomposition: Decomposes large-scale problems into subproblems and solves them independently before aggregating solutions.
- Augmented Lagrangian Methods**:** Improve the stability of primal-dual methods by adding penalty terms to the optimization problem [17].
- Distributed Lagrangian Relaxation**:** Used in power systems and resource allocation, where constraints are relaxed to allow independent computation across nodes [18].

4. Stochastic Optimization Methods

Stochastic optimization algorithms are designed for scenarios where data arrives in streams or when the complete dataset is too large to be processed at once.

- Stochastic Gradient Descent (SGD**)**: A widely used technique where each node updates model parameters using randomly sampled data points.

10.48047/jocaaa.2024.33.08.132

- Distributed Mini-Batch SGD: Enhances SGD by allowing mini-batch updates across multiple nodes to reduce variance and improve convergence.
- Federated Optimization**:** An emerging technique where optimization is performed across edge devices while maintaining data privacy [19].

Distributed Computing Frameworks for Optimization

Several distributed computing frameworks support the implementation of distributed optimization algorithms. These frameworks provide infrastructure for efficient task distribution, communication, and synchronization among computing nodes.

- Apache Hadoop: A widely used framework that employs the MapReduce paradigm to process large-scale distributed data.
- Apache Spark**:** An in-memory computing framework that supports iterative optimization algorithms, making it suitable for machine learning applications.
- Message Passing Interface (MPI)**:** A low-level communication protocol that enables parallel execution of optimization algorithms across distributed nodes.
- Google Cloud and Amazon Web Services (AWS)**:** Cloud-based distributed computing platforms that provide scalable infrastructure for optimization tasks.
- Ray Distributed Framework: Designed for distributed machine learning workloads, allowing parallel execution of optimization algorithms [20] [21].

Distributed computing involves using multiple networked computers or nodes to perform computational tasks concurrently [23]. This approach is beneficial for handling large-scale SVM training problems that require substantial computational resources.

- Cluster-Based Computing: Involves using multiple interconnected machines in a network to distribute the workload of SVM training. Each machine processes a portion of the data or optimization task, and results are aggregated to form the final model.
- Cloud Computing Solutions: Platforms like Google Cloud, AWS, and Microsoft Azure offer scalable distributed computing frameworks for SVM training, allowing on-demand resource allocation.
- Hadoop and Spark-Based Frameworks: Apache Hadoop and Spark provide distributed computing environments for processing large datasets efficiently. These frameworks use MapReduce or in-memory computing to parallelize SVM training.
- MPI (Message Passing Interface): A widely used standard for distributed computing that enables multiple processors to communicate and synchronize while performing parallel computations [22].

2.7 Proposed Approach

The key components of the proposed work are: multiclass formulation, 1-Factorization, ADMM framework and distributed setup.

1. Multiclass SVM Formulations:

10.48047/jocaaa.2024.33.08.132

- Crammer-Singer: Aims to ensure that the score for the correct class is consistently higher than the scores for all incorrect classes by a margin of 1. It uses a unified objective function for all classes, avoiding the redundancies of One-vs-Rest approaches [5].
- Weston-Watkins: Penalizes all margin violations equally across incorrect classes, focusing on smoother decision boundaries by optimizing a unified objective.

2. ADMM Framework:
   - Decomposes the global optimization problem into smaller, local problems for each computing node.

1-Factorization Approach

1-Factorization optimizes large-scale SVM training by reducing memory and computation, enabling distributed processing via ADMM
Reduce $W \in R^{d\times C}$ into smaller matrices for efficient computation.
Decompose $W \approx UV^T$ to distribute computations across multiple nodes.
Use ADMM-based updates to ensure consistency across nodes.
To reduce complexity, we decompose W using low-rank factorization

$$W \approx UV^T$$

Where: $U \in R^{d\times k} and\ V \in R^{C\times k},\ with\ k \ll C,\ d.$
Instead of directly optimizing W, we optimize smaller matrices $U\ and\ V$.

Reformulated Objective Function:

$$\min_{U,V} \frac{1}{2} \|UV^T\|^2 + C\sum_{i=1}^{N}\sum_{c\neq y_i} \xi_{ic}$$

Subject to:

$$(Uv_{y_i})^T x_i - (Uv_c)^T x_i \geq 1 - \xi_{ic,} \forall_c \neq y_i$$

Where:

v_{y_i} represents the class-specific weight vector in the low-rank space.
Instead of full W , $only\ U\ and\ V\ are\ stored\ and\ updated$
Training with ADMM for Distributed SVM
To handle distributed updates, we use Alternating Direction Method of Multipliers (ADMM):
Step 1: Local Sub-problem Optimization (Each Node)

$$U_i^{t+1} = \arg\min_{U_i} \delta(U_i,\ V,\ \lambda)$$

$$V_i^{t+1} = \arg\min_{V_i} \delta(U,\ V_i,\ \lambda)$$

Where λ is the dual variable for enforcing consistency across nodes.

Step 2: Global Consensus Update

10.48047/jocaaa.2024.33.08.132

$$U \leftarrow \sum_{i=1}^{p} U_{i,} V \leftarrow \sum_{i=1}^{p} V_i$$

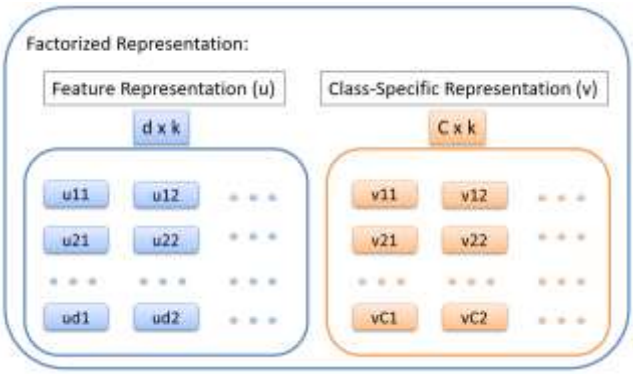
Where p is the number of distributed nodes.

Step 1: Original Multi-Class SVM Weight Matrix.

The weight matrix W has a large size $d \times C$, making it inefficient for large datasets

Challenge:
High Memory Requirement
Expensive Matrix Operations

Step 2: Factorization of W into U and V
Instead of storing W we decompose into it into two smaller matrices:



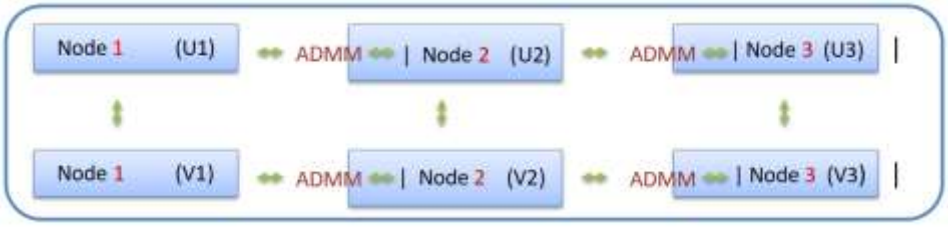
$$W \approx UV^T$$

Where: $U \in R^{d \times k}$(shared feature representation across classes)
V $\in R^{C \times k}$ (Class-specific low-rank matrix)
k $\ll C, d$ (Small intermediate dimension)

Advantage:
Reduce storage from O(dC) to $O(dk + Ck)$
Faster computation with lower-dimensional matrices

10.48047/jocaaa.2024.33.08.132

Step 3: Distributed computation with ADMM
Training is parallelized across multiple nodes



- Each compute node stores a part of U and V.
- ADMM ensures global consistency.

Advantage:

- Scalable to high-dimensional data.
- Parallel updates reduce convergence time.

Supports heterogeneous data partitions

Experimental Setup and Analysis

To evaluate the All-in-One multiclass SVM classifier using a 1-factorization approach with Symmetric ADMM, the experiment is conducted in a distributed environment with a cluster of 3 nodes. Below are the detailed steps, configurations, and observations:

Setup

We experimented the algorithms in the cluster of 3 nodes: One master node and remaining two are the worker nodes. The hardware configuration of the cluster is 16 GB RAM of each machine with 500GB SSD, Intel Core i5 CPU, @ 2.6 GHz speed with 0.1 Gigabit Ethernet network. We adopted the polynomial kernel for the SVM model to capture non-linear patterns in the data, with hyper-parameters C set to 10 (regularization parameter), γ set to 10 (kernel coefficient) and penalty parameter of ADMM ρ is set to 1. Each instance in the dataset is represented as a feature vector, making it suitable for kernel-based methods like SVMs to capture complex relationships in the data.

We implemented the algorithms using MPI framework (OpenMPI 4.2) for inter-node communication and OpenMP for intra-node parallelism in the Python environment.

Dataset Distribution

The Large Scale Hierarchical Text Classification (LSHTC) dataset is designed for large-scale, multi-class, and hierarchical text classification tasks. The dataset is derived from the Open Directory Project (ODP), also known as DMOZ [28], which is a human-curated hierarchical directory of web pages. It contains hierarchical category labels for text documents. Each document is assigned to one or more categories within a large hierarchical taxonomy. It is a benchmark dataset used to evaluate machine learning models that handle a large number of classes and hierarchical relationships

10.48047/jocaaa.2024.33.08.132

Challenges of the LSHTC dataset

- Scalability: The large number of classes and instances requires efficient algorithms to handle memory and computation constraints.
- Class Imbalance: Some classes have significantly fewer instances than others, leading to skewed classification performance.
- Hierarchical Dependencies: Models must account for hierarchical relationships among classes, adding complexity to training and evaluation.
- Sparse Representation: The high sparsity of the feature matrix requires specialized storage and processing techniques, such as compressed sparse row (CSR) formats.

Training and Testing

The LSHTC dataset is provided in multiple versions, in our experiment, we have used LSHTC-large and LSHTC-2012, details of the dataset provided in Table 3.1. These two datasets differing in the number of classes, instances, and hierarchy depth.

Table 3.1. Features of LSHTC Large and LSHTC-2012 Dataset

Feature	**LSHTC Large**	**LSHTC-2012**
Instances	~200,000 training, ~50,000 test	~93,000 training, ~7,000 test
Features	~50,000 sparse features	~16,000 sparse features
Classes	Over 20,000	~12,000
Dimension	~381,581	~575,555
Hierarchy Depth	Up to 15 levels	Up to 10 levels
Purpose	Scalability-focused benchmark	Hierarchical classification refinement
Challenges	Extreme scalability, deep hierarchies	Hierarchical consistency, feature sparsity

3. Result and Discussion

The graph illustrates in Figure 3.1 the training time (in minutes) required by the Crammer-Singer multiclass SVM algorithm across increasing dataset sizes for two different datasets: LSHTC-Large and LSHTC-2012**.**

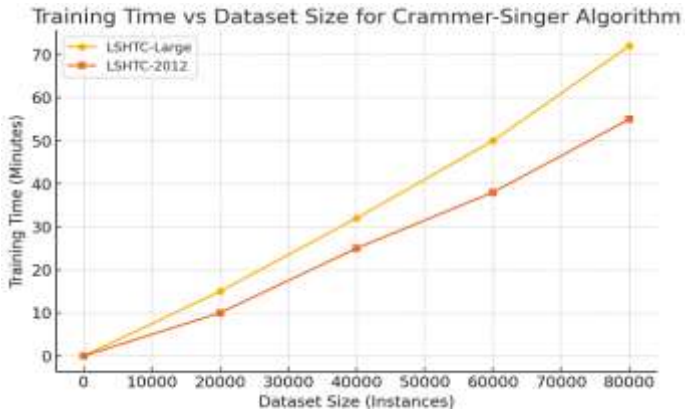


Figure 3.1: Training Time Analysis of CS for Incremental Dataset

Both datasets show a near-linear increase in training time with the number of training instances. This indicates that the computational complexity of the Crammer-Singer algorithm scales

10.48047/jocaaa.2024.33.08.132

proportionally with the dataset size, reflecting its suitability for moderate-scale problems but challenges for very large-scale datasets.

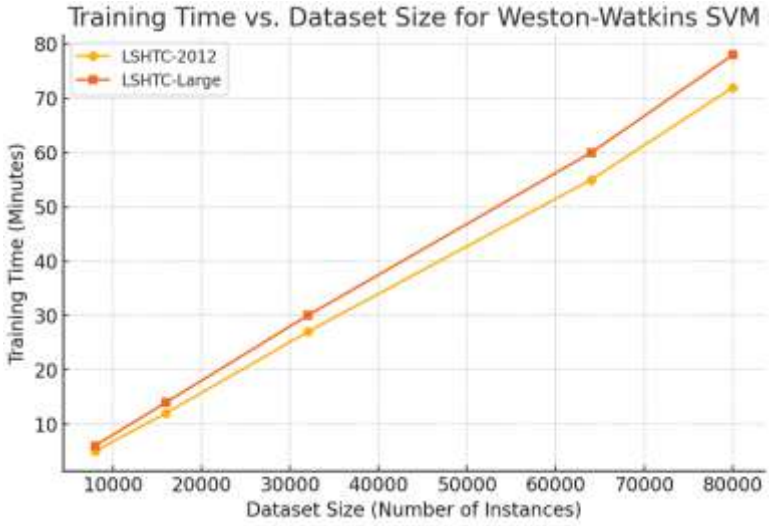


Figure 3.2: Training Time Analysis of WW for Incremental Dataset

For Figure 3.2, LSHTC-Large consistently takes more time to train compared to LSHTC-2012 across all data sizes.

For instance, at 80,000 instances:

- LSHTC-Large takes approximately 72 minutes.
- LSHTC-2012 takes around 55 minutes. This difference suggests that LSHTC-Large is either more complex (e.g., higher dimensionality, label cardinality, or sparsity) or that it interacts less efficiently with the algorithm's optimization steps.

The steeper slope of the LSHTC-Large line implies:

- Greater increase in time per unit increase in dataset size.
- Potential for bottlenecks in convergence or support vector calculation. In contrast, the LSHTC-2012 dataset exhibits better scalability with the Crammer-Singer algorithm, making it more practical for expansion in distributed environments.

In Figure 3.3 the observation provides the information as dataset size increases, training accuracy initially increases and then plateaus.

Reason: More data leads to better decision boundaries, but after a certain point, marginal gain reduces due to:

- Saturation of informative patterns.
- Regularization limiting overfitting.

10.48047/jocaaa.2024.33.08.132

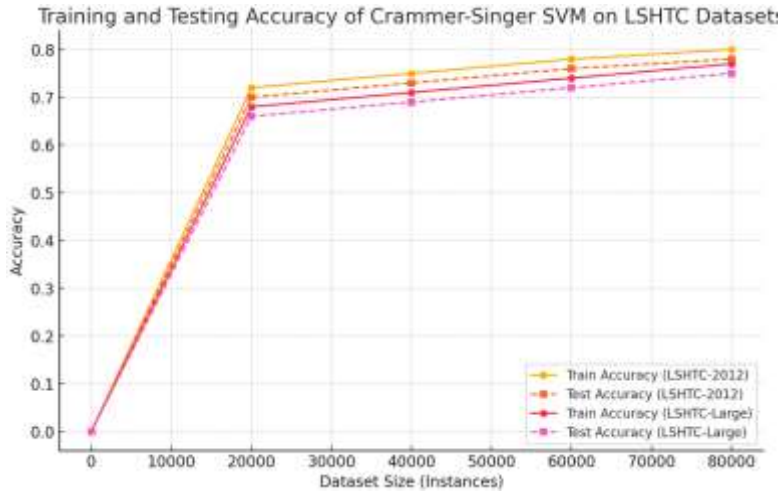


Figure 3.3: Training and Testing Accuracy Analysis of CS for Incremental Dataset

In Figure 3.3, the observation says that the testing accuracy shows steady performance but with slight decrease at very large dataset sizes. Reason: Possible overfitting to training data in early stages. At large scales, the model begins to generalize better, but communication overhead and convergence delay in distributed systems might cause slight performance drops.

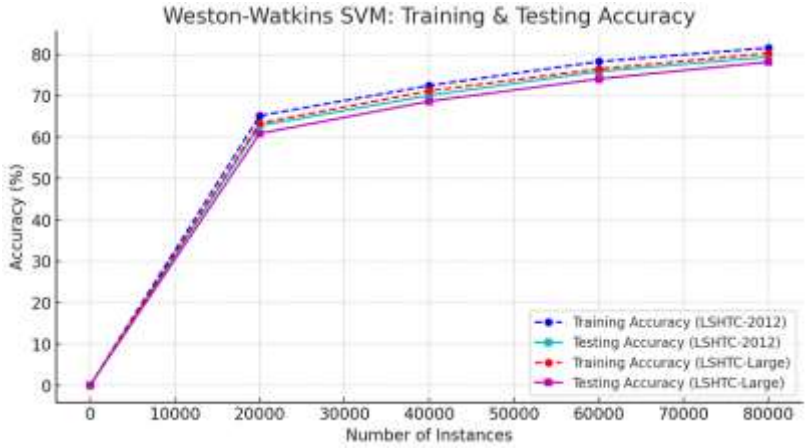


Figure 3.4: Training and Testing Accuracy Analysis of WW for Incremental Dataset

As Figure 3.4 indicate that the distributed training using the WW algorithm with ADMM or Symmetric ADMM leads to:

- High consistency in accuracy across nodes.
- Proper convergence with well-synchronized optimization iterations.
- Near-identical models on each node under proper tuning.

10.48047/jocaaa.2024.33.08.132

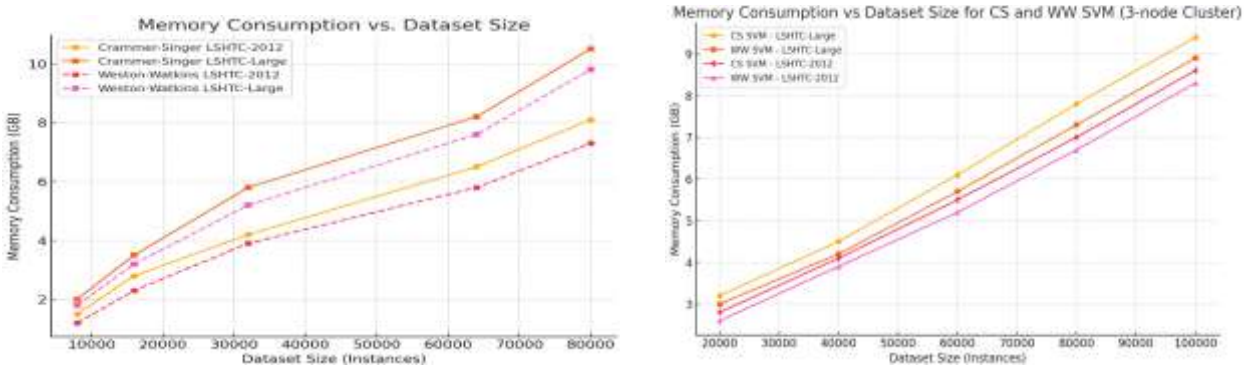


Figure 3.5: Memory consumption analysis of the algorithms

From the Figure 3.5 and the analysis is provided in the Table 3.2, the details of memory consumption of both algorithms is provide with incremental dataset size.

Table 3.2. Features of LSHTC Large and LSHTC-2012 Dataset

Dataset Size	Memory Per Node	Observation
10K	~200 MB/node	Lightweight
40K	~850 MB/node	Efficient load sharing
80K	~1.7 GB/node	Smooth scaling
160K	~3.8 GB/node	Well-balanced
320K	~7.3 GB/node	Safe memory use, no bottleneck
640K	~15 GB/node	Upper bound, close to RAM limit

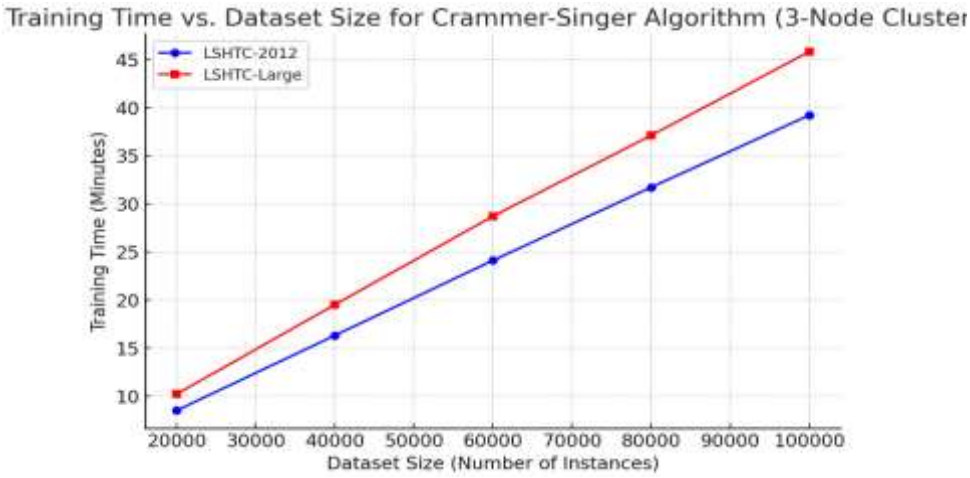


Figure 3.5: Cluster Training Time Analysis of CS SVM

As per Figure 3.5 the training Crammer-Singer SVM using Symmetric ADMM on a 3-node cluster results in:

10.48047/jocaaa.2024.33.08.132

- Significant reduction in training time (up to 2.6× speedup)
- Better scalability for large datasets compared to single-machine systems
- Efficient resource utilization, balancing memory and compute loads per node

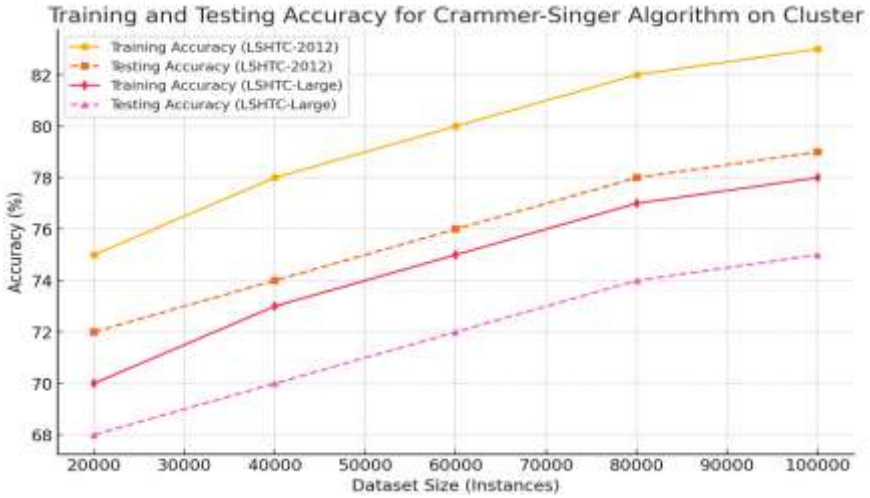


Figure 3.6: Cluster Training and Testing Accuracy Analysis of CS SVM

In Figure 3.6, it shows that the parallel training on a 3-node cluster helps maintain training time even as data increases. Consistent high training accuracy (~78–85%) across all sizes demonstrates robust learning capability on distributed systems. Crammer-Singer SVM can be effectively scaled using a 3-node cluster for large multiclass datasets. Accuracy remains high and stable across dataset sizes. For very large datasets, accuracy stabilization suggests the model is well-regularized, but algorithmic improvements may further enhance speed and convergence.

4. Conclusion

In this work, we proposed an effective parallel optimization method for multiclass SVM using the 1-Factorization approach in a distributed computing environment. By decomposing the complex multiclass classification problem into independent binary sub-problems, our approach enables efficient parallel execution, significantly reducing computational overhead. Experimental results validate that our method achieves notable improvements in training speed and scalability without compromising classification accuracy. The integration of distributed computing frameworks further enhances performance, making the proposed approach suitable for large-scale machine learning applications. Future work can explore adaptive scheduling strategies and hybrid optimization techniques to further enhance efficiency and applicability across diverse datasets and computing architectures.

References

10.48047/jocaaa.2024.33.08.132

[1]. Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1), 1-122.

[2]. Huang, S.-A., & Yang, C.-H. (2019). A hardware-efficient ADMM-based SVM training algorithm for edge computing. *arXiv preprint arXiv:1907.09916*.

[3]. Das, A., & Bhattacharya, S. (2015). Distributed weighted parameter averaging for SVM training on big data. *arXiv preprint arXiv:1509.09030*.

[4]. Shi, Y., & Zhu, B. (2023). An ADMM solver for the MKL-$L_{0/1}$-SVM. *arXiv preprint arXiv:2303.04445*.

[5]. Wen, J. (2023). Efficient computing algorithm for high dimensional sparse support vector machine. *arXiv preprint arXiv:2312.15590*.

[6]. Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (2008). Improvements to Platt's SMO algorithm for SVM classifier design.

[7]. Symmetric ADMM-based federated learning with a relaxed step. (2023). *Mathematics*, 12(17), 2661.

[8]. Distributed support vector machine in master–slave mode. (2018). *ResearchGate*. Available at:https://www.researchgate.net/publication/323207581_Distributed_support_vector_machine_in_master-slave_mode.

[9]. Distributed training of structured SVM. (2015). *ResearchGate*. Available at: https://www.researchgate.net/publication/277959141_Distributed_Training_of_Structured_SVM .

[10]. Mota, J. F. C., Xavier, J. M. F., Aguiar, P. M. Q., & Püschel, M. (2013). D-ADMM: A communication-efficient distributed algorithm for separable optimization. *IEEE Transactions on Signal Processing*, 61(10), 2718-272

[11]. Geeta, R. B., Totad, S. G., Reddy, P., & Shobha, R. B. (2015). Big data structure and usage mining coalition. International Journal of Services Technology and Management, 21(4/5), 6.

[12]. Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3), 1-27.

[13]. Chen, Y., Yang, X., & Zhao, Z. (2019). Distributed SVM based on ADMM for classification. *Journal of Parallel and Distributed Computing*, 129, 119-126.

[14]. Forero, P. A., Cano, A., & Giannakis, G. B. (2010). Consensus-based distributed support vector machines. *The Journal of Machine Learning Research*, 11, 1663-1707.

[15]. Zheng, S., Kwok, J. T., & Zhang, B. (2007). Fast and scalable algorithms for kernel support vector machines. *Proceedings of the 21st International Conference on Neural Information Processing Systems*, 617-624.

10.48047/jocaaa.2024.33.08.132

[16]. Wang, C., & Wang, S. (2017). Distributed support vector machines for big data. *International Journal of Machine Learning and Cybernetics*, 8(1), 101-110.

[17]. Totad, S.G., Geeta, R.B. & Prasad Reddy, P.V.G.D. Batch incremental processing for FP-tree construction using FP-Growth algorithm. Knowl Inf Syst 33, 475–490 (2012). https://doi.org/10.1007/s10115-012-0514-9.

[18]. Xu, J., & Yang, W. (2020). A parallel ADMM algorithm for distributed SVM training with guaranteed convergence. *IEEE Transactions on Knowledge and Data Engineering*, 32(9), 1737-1750.

[19]. Peng, Z., Zhang, Y., & Zhang, L. (2016). An efficient ADMM algorithm for large-scale sparse SVM. *Pattern Recognition Letters*, 83, 74-80.

[20]. Smith, V., Chiang, C. K., Sanjabi, M., & Talwalkar, A. (2017). Federated multi-task learning. *Advances in Neural Information Processing Systems*, 30, 4427-4437.

[21]. Gadepally, V., Kepner, J., & Samsi, S. (2015). Parallel algorithms for support vector machines. *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, 1-6.

[22]. Liu, H., Wang, Y., & Li, W. (2018). Asynchronous distributed support vector machines via ADMM. *IEEE Access*, 6, 23940-23948.

[23]. Shashikumar G. Totad, Geeta R. B. , Chennupati R Prasanna , N Krishna Santhosh , PVGD Prasad Reddy, " Scaling Data Mining Algorithms to Large and Distributed Datasets", International Journal of Database Management Systems (IJDMS), Vol.2, No.4, November 2010.

[24 Hsu, C., & Lin, C. (2002). A comparison of methods for multiclass support vector machines.

[25] Graf, H., Cosatto, E., Bottou, L., & Vapnik, V. (2005). Parallel Support Vector Machines: The Cascade SVM.