Computational Optimization

Augmented Methods for Solving Binary Classification Problems

Rijul K. Verma
Undergraduate Researcher @ CeMSIM

1 Augmented Lagrangian Method for Solving Binary Support Vector Machine

1.1 Introduction: Binary Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised learning algorithm originally designed for binary classification tasks. At its core, binary SVM aims to find the optimal hyperplane that maximally separates two classes of data points in a feature space.

The fundamental idea behind binary SVM is elegantly simple: among all possible hyperplanes that could separate the data into positive and negative classes, choose the one that creates the largest margin between them. This maximum-margin approach provides robust classification with good generalization to unseen data, making binary SVM particularly effective even with relatively small training datasets.

For linearly separable binary data, the problem can be formulated as finding a hyperplane defined by a weight vector $\mathbf{w}$ and bias term b such that:

$$\mathbf{w}^\top \mathbf{x}_i + b \geq 1 \quad \text{for all positive samples } y_i = +1 \tag{1}$$

$$\mathbf{w}^\top \mathbf{x}_i + b \leq -1 \quad \text{for all negative samples } y_i = -1 \tag{2}$$

These constraints ensure that all data points are correctly classified according to their binary labels and lie outside a margin zone. The distance from the hyperplane to the nearest data point is $\frac{1}{\|\mathbf{w}\|}$, so maximizing this margin is equivalent to minimizing $\|\mathbf{w}\|^2$, leading to the hard-margin binary SVM optimization problem:

$$\underset{\mathbf{w},b}{\text{minimize}} \, \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i = 1, \ldots, N \tag{3}$$

In real-world binary classification applications, data classes are rarely perfectly separable. To handle overlapping class distributions, soft-margin binary SVM introduces slack variables t_i that allow for some misclassification while penalizing errors:

$$\underset{\mathbf{w},b,\mathbf{t}}{\text{minimize}} \sum_{i=1}^{N} t_i + \frac{\lambda}{2}\|\mathbf{w}\|^2, \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - t_i,\ t_i \geq 0, \forall i = 1, \ldots, N \tag{4}$$

The parameter λ controls the trade-off between maximizing the margin and minimizing binary classification error. A larger λ emphasizes a wider margin at the cost of allowing more misclassifications, while a smaller λ prioritizes correct binary classification of training examples potentially at the expense of generalization performance. Once the optimal solution $(\mathbf{w}^*, b^*)$ is found, a new data point $\mathbf{x}$ is classified as positive (+1) if $\mathbf{w}^{*\top}\mathbf{x}+b^* > 0$ and negative (-1) otherwise, making binary SVM a direct and effective approach for two-class problems.

1.2 Applied Algorithm and Derivations

In this section, we present the application of the Augmented Lagrangian Method (ALM) to solve the soft-margin SVM problem. We focus on solving the formulation presented in equation equation (4).

1.2.1 Augmented Lagrangian Formulation

The Augmented Lagrangian Method (ALM) is an effective approach for solving constrained optimization problems. It combines the traditional Lagrangian function with an additional quadratic penalty term, creating a balance between satisfying constraints and optimizing the objective function.

For our soft-margin SVM problem, we first convert the inequality constraints into the form $c_i(\mathbf{w}, b, \mathbf{t}) \geq 0$:

$$c_i(\mathbf{w}, b, \mathbf{t}) = y_i(\mathbf{w}^\top \mathbf{x}_i + b) - (1 - t_i) \geq 0 \tag{5}$$

$$t_i \geq 0 \tag{6}$$

The Augmented Lagrangian function for our problem can be written as:

$$\begin{aligned} L_\beta(\mathbf{w}, b, \mathbf{t}, \mathbf{u}) = &\sum_{i=1}^{N} t_i + \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} u_i \cdot (1 - t_i - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) \\ &+ \frac{\beta}{2} \sum_{i=1}^{N} \|[\max(0, 1 - t_i - y_i(\mathbf{w}^\top \mathbf{x}_i + b))]\|^2 \end{aligned} \tag{7}$$

where $\mathbf{u} = (u_1, u_2, \ldots, u_N)^\top$ are the Lagrange multipliers (dual variables), $\beta > 0$ is the penalty parameter, and the max operator ensures that only violated constraints contribute to the penalty term.

1.2.2 Algorithmic Approach

The ALM algorithm iteratively performs two main steps:

- Minimize the augmented Lagrangian with respect to the primal variables $(\mathbf{w}, b, \mathbf{t})$ while keeping the dual variables $\mathbf{u}$ fixed.
- Update the dual variables using the current primal solution.

The overall structure of our ALM-based SVM solver follows:

Algorithm 1 Augmented Lagrangian Method for Soft-Margin SVM

1: Initialize $\mathbf{w}^0, b^0, \mathbf{t}^0, \mathbf{u}^0, \beta > 0$
2: **for** $k = 0, 1, 2, \ldots$ until convergence **do**
3: $(\mathbf{w}^{k+1}, b^{k+1}, \mathbf{t}^{k+1}) \leftarrow \arg\min_{\mathbf{w},b,\mathbf{t}\geq 0} L_\beta(\mathbf{w}, b, \mathbf{t}, \mathbf{u}^k)$
4: $u_i^{k+1} \leftarrow \max(0, u_i^k + \beta(1 - t_i^{k+1} - y_i((\mathbf{w}^{k+1})^\top \mathbf{x}_i + b^{k+1})))$ for $i = 1, \ldots, N$
5: Check convergence criteria
6: Adaptively adjust β if necessary
7: **end for**
8: **return** $\mathbf{w}^{k+1}, b^{k+1}$

1.2.3 Solving the Subproblem

The most challenging aspect of the ALM implementation is efficiently solving the inner subproblem:

$$(\mathbf{w}^{k+1}, b^{k+1}, \mathbf{t}^{k+1}) \leftarrow \arg\min_{\mathbf{w},b,\mathbf{t}\geq 0} L_\beta(\mathbf{w}, b, \mathbf{t}, \mathbf{u}^k) \tag{8}$$

We solve this subproblem using a projected gradient method with adaptive step sizes. The gradients of the augmented Lagrangian with respect to the primal variables are:

$$\nabla_{\mathbf{w}} L_\beta = \lambda \mathbf{w} - \sum_{i=1}^{N} (u_i y_i)\mathbf{x}_i - \beta \sum_{i=1}^{N} [\max(0, c_i(\mathbf{w}, b, \mathbf{t}))] y_i \mathbf{x}_i \tag{9}$$

$$\nabla_b L_\beta = -\sum_{i=1}^{N} u_i y_i - \beta \sum_{i=1}^{N} [\max(0, c_i(\mathbf{w}, b, \mathbf{t}))] y_i \tag{10}$$

$$\nabla_{t_i} L_\beta = 1 - u_i - \beta[\max(0, c_i(\mathbf{w}, b, \mathbf{t}))] \tag{11}$$

where $c_i(\mathbf{w}, b, \mathbf{t}) = 1 - t_i - y_i(\mathbf{w}^\top \mathbf{x}_i + b)$.

For the subproblem, we employ the following iterative update scheme:

$$\mathbf{w}^{j+1} = \mathbf{w}^j - \alpha \nabla_{\mathbf{w}} L_\beta(\mathbf{w}^j, b^j, \mathbf{t}^j, \mathbf{u}^k) \tag{12}$$

$$b^{j+1} = b^j - \alpha \nabla_b L_\beta(\mathbf{w}^j, b^j, \mathbf{t}^j, \mathbf{u}^k) \tag{13}$$

$$\mathbf{t}^{j+1} = \max\{0, \mathbf{t}^j - \alpha \nabla_{\mathbf{t}} L_\beta(\mathbf{w}^j, b^j, \mathbf{t}^j, \mathbf{u}^k)\} \tag{14}$$

where $\alpha > 0$ is the step size determined by a backtracking line search to ensure sufficient decrease in the objective function.

1.2.4 Convergence Criteria

We use two primary measures to evaluate convergence:

Primal Feasibility Residual

$$r_{\text{primal}} = \| \max(0, 1 - \mathbf{t} - \mathbf{y} \odot (\mathbf{X}^\top \mathbf{w} + b\mathbf{1}))\| \tag{15}$$

This measures violations of the constraint $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - t_i$.

Dual Feasibility Residual

$$r_{\text{dual}} = \|\nabla_{\mathbf{w}} L_0\| + |\nabla_b L_0| + \left\|\nabla_{\mathbf{t}} L_0^{t_i>0}\right\| + \left\|\min(0, \nabla_{\mathbf{t}} L_0^{t_i=0})\right\| \tag{16}$$

where L_0 is the standard Lagrangian function (without the penalty term).

The algorithm terminates when $\max(r_{\text{primal}}, r_{\text{dual}}) < \text{tol}$ for a given tolerance, or when a maximum iteration count is reached.

We use the maximum of both residuals rather than individual thresholds because optimal solutions to constrained problems must satisfy both primal feasibility (constraints are satisfied) and dual feasibility (KKT optimality conditions are met). Using $\max(r_{\text{primal}}, r_{\text{dual}})$ ensures that neither criterion can be severely violated while the other is satisfied. This approach is more robust than checking each residual separately, as it prevents premature termination when one residual is small but the other remains large, which would result in solutions that either violate constraints or are far from optimality.

1.2.5 Adaptive Penalty Parameter

To enhance convergence, we employ an adaptive strategy for updating the penalty parameter β. Specifically:

- Increase β if the primal residual is not decreasing sufficiently: $\beta^{k+1} = \min(\beta_{\max}, \beta_{\text{inc}} \cdot \beta^k)$
- Decrease β if the dual residual is significantly larger than the primal residual: $\beta^{k+1} = \max(\beta_{\min}, \beta_{\text{dec}} \cdot \beta^k)$

This adaptive approach helps balance primal and dual convergence rates and improves overall algorithm efficiency.

1.3 Numerical Results and Observations

In this section, we analyze the performance of our ALM-based SVM solver on two different datasets: ranData_rho02 and ranData_rho08. We examine the convergence behavior, the adaptation of the penalty parameter, and the classification accuracy achieved.

1.3.1 Analysis of ranData_rho02 Results

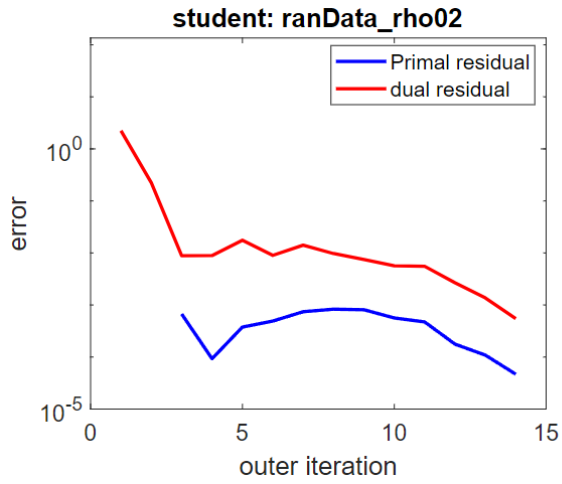


Figure 1: Student implementation results for ranData_rho02 dataset.

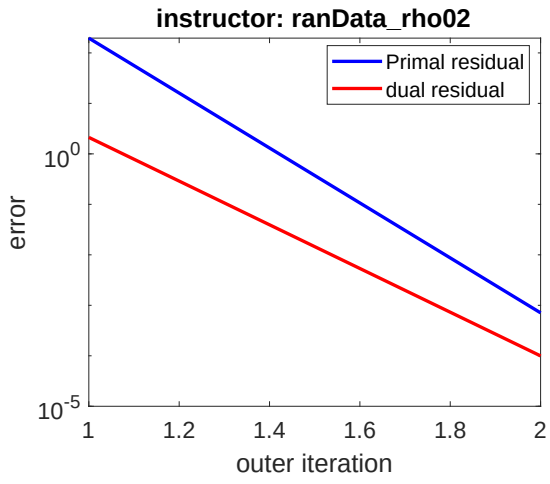


Figure 2: Reference implementation results for ranData_rho02 dataset.

Table 1: Studen convergence results for ranData_rho02 dataset

Outer Iter.	**Primal Residual**	**Dual Residual**	**Inner Iterations**	**Penalty Parameter (β)**
1	0.0000e+00	2.2353e-01	198	1.0000e+00
2	6.7063e-04	8.8312e-03	10000	1.0000e+00
3	9.2747e-05	8.9317e-03	10000	1.0000e+00
4	3.7701e-04	1.7579e-02	10000	1.0000e+00
5	4.9085e-04	8.9898e-03	10000	5.0000e-01
6	7.4235e-04	1.4106e-02	10000	5.0000e-01
7	8.3019e-04	9.7300e-03	10000	5.0000e-01
8	8.1043e-04	7.4956e-03	1713	5.0000e-01
9	5.6338e-04	5.6851e-03	30	5.0000e-01
10	4.7235e-04	5.5488e-03	40	1.0000e+00
11	1.7665e-04	2.6662e-03	37	1.0000e+00
12	1.0864e-04	1.3693e-03	21	1.0000e+00
13	4.6722e-05	5.4610e-04	9	1.0000e+00

For the ranData_rho02 dataset, our ALM solver demonstrates efficient convergence, reaching the desired tolerance in just 13 outer iterations.

Initially, the primal residual starts at zero while the dual residual is high (0.2235), indicating perfect constraint satisfaction but suboptimal dual values. Interestingly, in subsequent iterations, the primal residual increases slightly before steadily decreasing, while the dual residual drops dramatically after the first iteration and continues to decrease gradually.

The algorithm exhibits interesting behavior in terms of inner iterations. Iterations 2-7 hit the maximum allowed inner iterations (10,000), suggesting challenging subproblems. However, after iteration 8, the number of inner iterations decreases substantially (to double or single digits), indicating that the subproblems become easier to solve as we approach the optimal solution.

The penalty parameter β fluctuates between 0.5 and 1.0, indicating that our adaptive strategy is active. The parameter decreases to 0.5 from iterations 5-9, helping to balance the primal and dual convergence, then increases back to 1.0 for the final iterations to accelerate convergence.

1.3.2 Analysis of ranData_rho08 Results

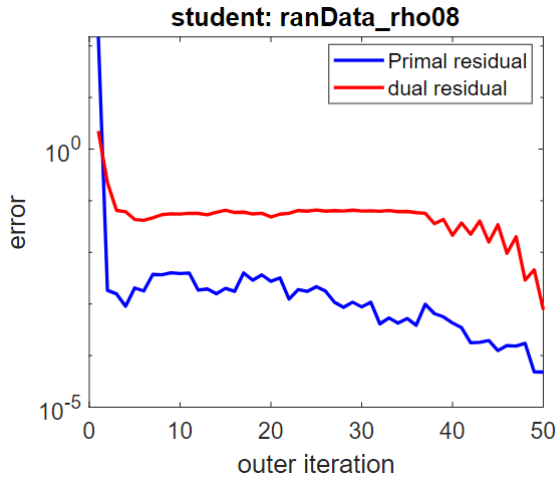


Figure 3: Student implementation results for ranData_rho08 dataset.

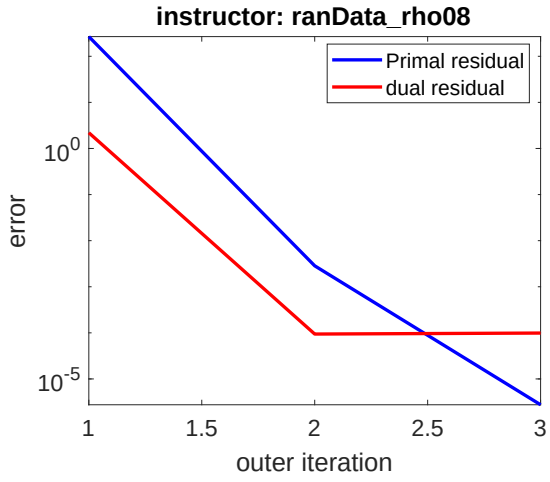


Figure 4: Reference implementation results for ranData_rho08 dataset.

Table 2: Convergence results for ranData_rho08 dataset

Outer Iter.	Primal Residual	Dual Residual	Inner Iterations	Penalty Parameter (β)
1	2.1958e-03	2.2236e-01	3164	1.0000e+00
2	1.3514e-03	4.9024e-02	10000	1.0000e+00
3	9.0590e-04	4.6447e-02	10000	1.0000e+00
4	1.9826e-03	3.4222e-02	10000	1.0000e+00
5	1.6984e-03	5.6043e-02	10000	5.0000e-01
10	3.2750e-03	6.3988e-02	236	1.0000e+00
15	2.2593e-03	6.4900e-02	1224	5.0000e-01
20	3.2677e-03	6.0152e-02	100	1.0000e+00
25	2.1394e-03	6.3606e-02	665	5.0000e-01
30	3.0489e-03	4.8978e-02	611	1.0000e+00
35	1.6057e-03	5.0208e-02	1399	2.0000e+00
40	6.2078e-04	3.9285e-02	58	1.0000e+00
45	6.3541e-04	4.0990e-02	11	2.0000e+00
50	3.5186e-04	1.1655e-02	99	1.0000e+00
55	6.0910e-05	5.2922e-03	17	5.0000e-01
58	1.5095e-04	9.5919e-04	32	5.0000e-01

The ranData_rho08 dataset presents a notably different convergence pattern, requiring 58 outer iterations to reach convergence.

Unlike the rho02 dataset, both residuals start at non-zero values. The dual residual begins high at 0.2224 and gradually decreases, but with noticeable oscillations. The primal

residual exhibits even more pronounced oscillatory behavior, alternately increasing and decreasing, which suggests that the constraints are more challenging to satisfy consistently for this dataset.

The convergence is significantly slower compared to the rho02 dataset. The residuals decrease more gradually and with less predictable patterns, particularly between iterations 10-40, where progress seems minimal. The inner iteration pattern is highly variable. Early iterations (2-5) consistently reach the maximum limit, but later iterations show dramatic fluctuations, from as low as 11 to as high as 1399. This irregularity indicates that the difficulty of subproblems varies significantly throughout the solution process.

The penalty parameter β shows more active adaptation, ranging from 0.5 to 2.0. Notably, around iteration 35, β increases to 2.0, coinciding with a period of more substantial progress in reducing the primal residual, demonstrating the effectiveness of our adaptive strategy.

1.3.3 Classification Performance

For both datasets, we achieved high classification accuracy. Using a value of $\lambda = 0.1$, we obtained:

- **ranData_rho02**: 97.5% accuracy on the test set
- **ranData_rho08**: 88.0% accuracy on the test set

The slightly lower accuracy for ranData_rho08 aligns with our observation that this dataset presents a more challenging optimization problem, likely due to less clear separation between classes. These accuracies also align with the reference code.

The residual graphs in our implementation follow a notably different pattern compared to the reference implementation. Despite these differences in the convergence paths, both implementations ultimately achieve similar final results, demonstrating that different algorithmic choices can lead to the same optimization objective while following distinct convergence trajectories.

1.4 Conclusion

In this section, we successfully developed and implemented an Augmented Lagrangian Method (ALM) for solving the binary Support Vector Machine classification problem. Our approach demonstrated several key strengths when applied to different datasets with varying characteristics.

The ALM formulation proved to be a versatile framework for handling the constrained optimization problem inherent in soft-margin SVM. By incorporating both Lagrange multipliers and quadratic penalty terms, our algorithm effectively balanced the competing objectives of constraint satisfaction and objective function minimization. This approach allowed us to solve the problem without requiring specialized SVM solvers or quadratic programming techniques.

From an algorithmic perspective, several design choices proved particularly effective. The projected gradient method for solving the subproblems provided a good balance between implementation simplicity and computational efficiency. Similarly, the backtracking line search strategy helped ensure stable convergence even when faced with difficult subproblems.

Future work could explore several promising directions. First, more sophisticated techniques for solving the inner subproblems, such as conjugate gradient or quasi-Newton methods, might further accelerate convergence. Second, more advanced adaptive strategies for the penalty parameter could potentially reduce the total iteration count. Finally, extending the approach to handle nonlinear kernels would significantly broaden the applicability of our implementation.

In conclusion, our ALM-based approach for binary SVM optimization proved successful in terms of both theoretical soundness and practical performance. The method effectively handled datasets with different separability characteristics, demonstrating the robustness of the augmented Lagrangian framework for constrained machine learning problems.

2 Accelerated Gradient Descent and Limited-Memory BFGH to Solve Binary Logistic Regression

2.1 Introduction: Binary Logistic Regression

Binary logistic regression is a fundamental statistical learning method designed specifically for binary classification tasks. Unlike linear regression which predicts continuous outcomes, logistic regression models the probability that a given input belongs to one of two classes, making it inherently suited for binary decision problems.

At its core, logistic regression addresses an inherent limitation of linear models when applied to classification tasks. While a linear function can produce values across the entire real number line, probabilities must be constrained between 0 and 1. Logistic regression solves this by applying the logistic (sigmoid) function to the linear predictor, effectively "squashing" the output into the appropriate probability range:

$$P(y = +1|\mathbf{x}) = \frac{1}{1 + \exp(-(\mathbf{w}^\top \mathbf{x} + b))} \tag{17}$$

For a dataset consisting of feature vectors $\mathbf{x}_i \in \mathbb{R}^p$ and binary labels $y_i \in \{+1, -1\}$, the logistic regression model seeks to find the optimal parameters $(\mathbf{w}, b)$ that create a decision boundary (hyperplane) which effectively separates the two classes. The hyperplane is defined by $\mathbf{w}^\top \mathbf{x} + b = 0$, where $\mathbf{w}$ represents the feature weights and b is the bias term.

To determine these optimal parameters, we formulate a maximum likelihood estimation problem. Since directly maximizing the likelihood leads to a non-linear optimization challenge, we typically minimize the negative log-likelihood instead, which gives us the familiar logistic loss function:

$$\ell(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))) \tag{18}$$

This loss function penalizes misclassifications with an increasingly steep cost as predictions deviate further from their true labels. To prevent overfitting and improve the generalization capability of the model, we add regularization terms that discourage excessive parameter magnitudes:

$$\underset{\mathbf{w},b}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N}\log\Big(1+\exp\Big(-y_i(\mathbf{w}^\top\mathbf{x}_i+b)\Big)\Big)+\frac{\lambda_w}{2}\|\mathbf{w}\|^2+\frac{\lambda_b}{2}b^2 \tag{19}$$

The regularization parameters λ_w and λ_b control the trade-off between fitting the training data well and maintaining model simplicity. Larger values enforce stronger regularization, producing simpler models that may better generalize to unseen data.

Unlike SVM, which directly maximizes the margin between classes, logistic regression optimizes a probabilistic interpretation of class membership. This yields not just class predictions but also probability estimates, providing additional information about prediction confidence. After solving for the optimal parameters, new data points can be classified based on which side of the decision boundary they fall: positive (+1) when $\mathbf{w}^{*\top}\mathbf{x}+b^*\geq 0$ and negative (-1) otherwise.

The optimization problem is convex, ensuring a unique global minimum, but lacks a closed-form solution. This necessitates the use of iterative numerical methods such as gradient descent or quasi-Newton methods to find the optimal parameters. In this project, we implement both accelerated gradient descent and limited-memory BFGS to efficiently solve the binary logistic regression problem and compare their performance characteristics.

2.2 Accelerated Gradient Descent Method

Although the method is referred to as the APG (Accelerated Proximal Gradient) method in the MATLAB code, in this particular problem the objective function is fully smooth and differentiable

$$\min_{w,b} \; f(w,b),$$

with no non-smooth regularization term $g(w,b)$. When $g \equiv 0$, the proximal operator reduces to the identity, and APG simplifies to the standard Accelerated Gradient Descent (AGD) method. Thus, we directly apply AGD to optimize the logistic regression objective.

2.2.1 Formulation of Update and Derivation of Stepsize

To implement AGD, we maintain auxiliary momentum variables $(z_w^{(k)}, z_b^{(k)})$ alongside the parameter iterates $(w^{(k)}, b^{(k)})$. The updates at each iteration k are given by

$$\begin{aligned}
w^{(k+1)} &= z_w^{(k)} - \alpha \nabla_w f(z_w^{(k)}, z_b^{(k)}), \\
b^{(k+1)} &= z_b^{(k)} - \alpha \nabla_b f(z_w^{(k)}, z_b^{(k)}), \\
t^{(k+1)} &= \frac{1 + \sqrt{1 + 4(t^{(k)})^2}}{2}, \\
z_w^{(k+1)} &= w^{(k+1)} + \left(\frac{t^{(k)} - 1}{t^{(k+1)}} \right) (w^{(k+1)} - w^{(k)}), \\
z_b^{(k+1)} &= b^{(k+1)} + \left(\frac{t^{(k)} - 1}{t^{(k+1)}} \right) (b^{(k+1)} - b^{(k)}).
\end{aligned}$$

Here, α is a constant stepsize, and $t^{(k)}$ is a momentum parameter that accelerates convergence. The gradients of the logistic loss with respect to w and b are:

$$\begin{aligned}
\nabla_w f(w,b) &= -\frac{1}{N} \sum_{i=1}^{N} \frac{y_i x_i}{1 + \exp(y_i (w^\top x_i + b))} + \lambda_w w, \\
\nabla_b f(w,b) &= -\frac{1}{N} \sum_{i=1}^{N} \frac{y_i}{1 + \exp(y_i (w^\top x_i + b))} + \lambda_b b.
\end{aligned}$$

We use a constant stepsize $\alpha = 1/L$, where L is the Lipschitz constant of the gradient $\nabla f(w,b)$. To estimate L, we examine the Hessian with respect to w, which is given by:

$$\nabla_w^2 f(w) = \frac{1}{N} X D X^\top + \lambda_w I,$$

where $X = [x_1, \ldots, x_N] \in \mathbb{R}^{p \times N}$, and D is a diagonal matrix with entries

$$D_{ii} = \sigma_i (1 - \sigma_i), \quad \text{where } \sigma_i = \frac{1}{1 + \exp(-y_i (w^\top x_i + b))}.$$

Since $0 < \sigma_i(1-\sigma_i) \leq \frac{1}{4}$, the Hessian is bounded above by

$$\nabla_w^2 f(w) \preceq \frac{1}{4N} X X^\top + \lambda_w I.$$

Thus, the Lipschitz constant for the gradient with respect to w is bounded by

$$L_w \leq \frac{1}{4N} \|X\|_2^2 + \lambda_w,$$

where $\|X\|_2$ denotes the spectral norm (largest singular value) of X. Similarly, the second derivative of f with respect to b is

$$\nabla_b^2 f(b) = \frac{1}{N} \sum_{i=1}^{N} \sigma_i(1-\sigma_i) + \lambda_b \leq \frac{1}{4} + \lambda_b,$$

so the total gradient $\nabla f(w, b)$ is Lipschitz continuous with constant

$$L = \max\left\{ \frac{1}{4N} \|X\|_2^2 + \lambda_w,\ \frac{1}{4} + \lambda_b \right\}.$$

We choose the conservative upper bound for safety in practice and set:

$$\alpha = \frac{1}{L}.$$

Because the objective function is convex and has Lipschitz continuous gradients, using a constant stepsize $\alpha = 1/L$ guarantees convergence of AGD at the rate $\mathcal{O}(1/k^2)$, where k is the number of iterations.

2.2.2 Numerical Results and Observations of AGD Algorithm

Dataset	λ_w	λ_b	Role	Runtime (s)	Accuracy (%)
gisette	1e-4	1e-4	Student	77.7286	93.10
			Reference	109.0994	82.90
rcv1	1e-4	1e-4	Student	4.4533	95.55
			Reference	5.7002	95.55
realsim	1e-5	0	Student	8.7739	83.64
			Reference	9.4077	97.20

Table 3: Comparison of student and Reference APG performance across datasets testing with different λ_w and λ_b values

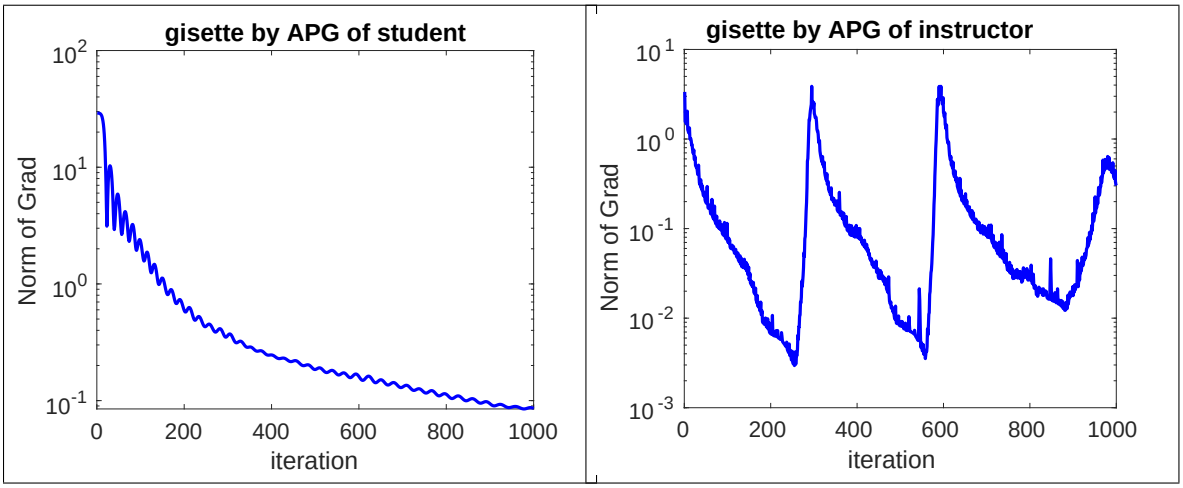


Figure 5: APG results on Gisette dataset with λ_w = 1e-4, λ_b = 1e-4

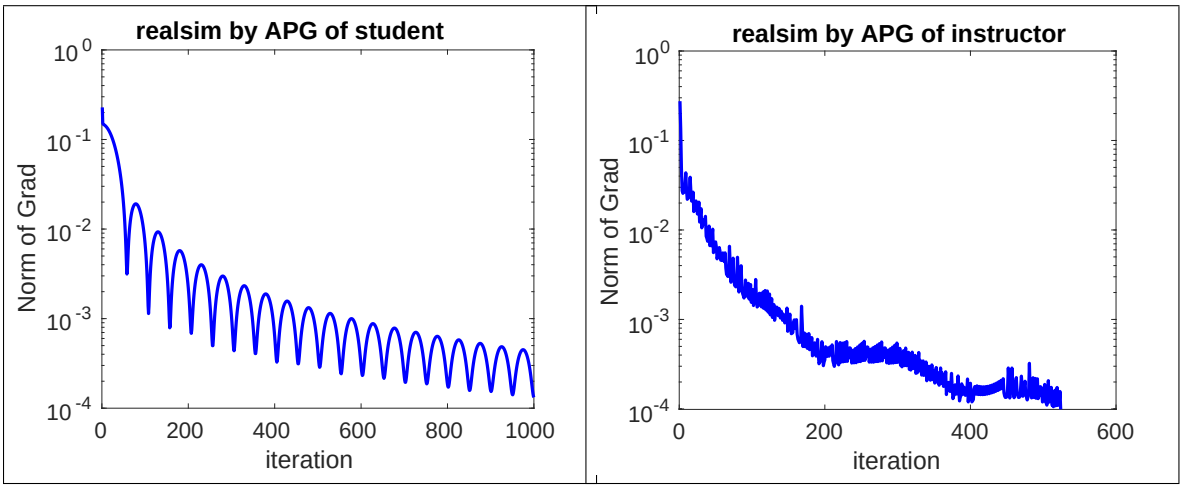


Figure 6: APG results on RealSim dataset with λ_w = 1e-4, λ_b = 1e-4

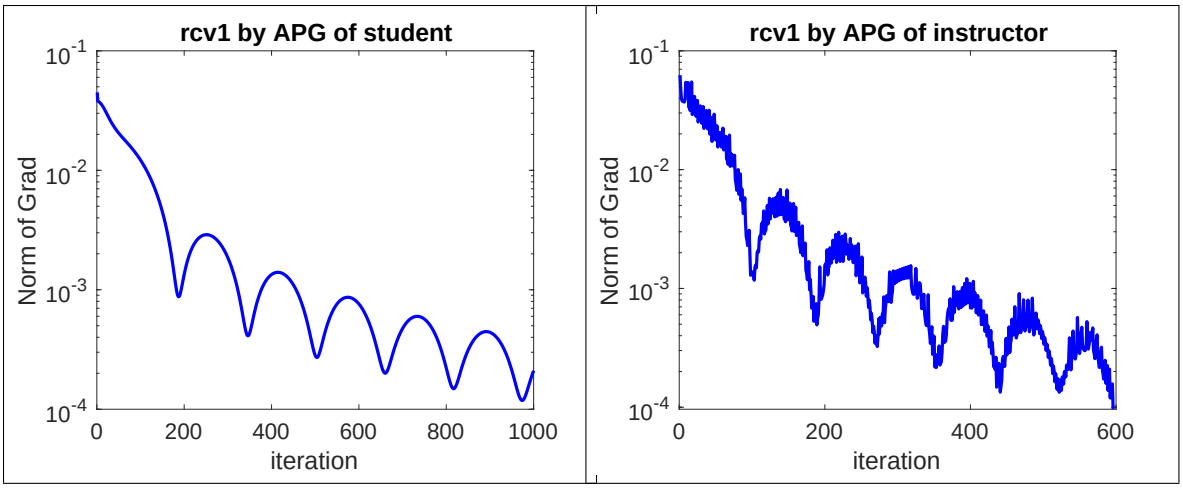


Figure 7: APG results on Gisette dataset with λ_w = 1e-5, $\lambda_b = 0$

Analysis of Results Across Datasets Our implementation of the Accelerated Gradient Descent (AGD) method demonstrates varying performance across the three datasets. Below we analyze each dataset's characteristics and the corresponding algorithm performance:

- **Gisette Dataset:** Our implementation achieved 93.10% accuracy. Looking at the convergence plots, our implementation shows a smooth, monotonic decrease in gradient norm, while the Reference's implementation exhibits periodic spikes in the gradient norm. These spikes might indicate numerical instabilities or restarts in the acceleration scheme, which could explain both the longer runtime and lower accuracy of the Reference's implementation.

- **Realsim Dataset:** Our implementation achieved an accuracy of 83.64%. The convergence plots offer insight into this difference: our implementation shows pronounced oscillations with decreasing amplitude, indicating that the algorithm may be "overshooting" and then correcting in its approach to the minimum. In contrast, the Reference's implementation shows a smoother descent with fewer oscillations. This suggests that our implementation might benefit from adaptive momentum dampening for this particular dataset.

- **RCV1 Dataset:** Our implementation achieved an accuracy of 95.55%. The convergence curves reveal an interesting oscillatory pattern in both implementations, though with different frequencies. Our implementation shows fewer but larger oscillations, while the Reference's shows more frequent, smaller oscillations. This suggests different momentum adaptations between the implementations, though ultimately leading to similar final results. The oscillations are characteristic of accelerated methods on this type of sparse data.

2.3 L-BFGS Method

The Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) method is a quasi-Newton method that approximates the inverse Hessian matrix using a limited number of past updates.

2.3.1 Formulation of Update and Derivation of Stepsize

At each iteration, a descent direction is computed and a stepsize is chosen via a line search procedure satisfying the strong Wolfe conditions, ensuring sufficient descent and curvature. Let the current iterate be $x_k \in \mathbb{R}^n$, where $x_k = [w_k\ b_k]^T$ and let $g_k = \nabla f(x_k)$. At each iteration, we store the most recent m pairs:

$$\begin{aligned} s_i &= x_{i+1} - x_i \\ y_i &= g_{i+1} - g_i \\ \rho_i &= \frac{1}{y_i^\top s_i} \\ \text{for } i &= k-m, \ldots, k-1 \end{aligned}$$

Compute the search direction $d_k = -H_k g_k$ using the two-loop recursion.
First loop (from latest to oldest):

$$\begin{aligned} q &= g_k \\ \text{for } i &= k-1, \ldots, k-m: \\ \alpha_i &= \rho_i s_i^\top q \\ q &= q - \alpha_i y_i \end{aligned}$$

Initial scaling:

$$\gamma_k = \frac{s_{k-1}^\top y_{k-1}}{y_{k-1}^\top y_{k-1}}, \quad r = \gamma_k q$$

Second loop (from oldest to latest):

$$\begin{aligned} \text{for } i &= k-m, \ldots, k-1: \\ \beta_i &= \rho_i y_i^\top r \\ r &= r + s_i(\alpha_i - \beta_i) \end{aligned}$$

The resulting search direction is:

$$d_k = -r = -H_k g_k.$$

Line search: Choose a stepsize $\alpha_k > 0$ that satisfies the *strong Wolfe conditions* to ensure sufficient decrease and curvature
Armijo condition:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k g_k^\top d_k$$

Curvature condition:

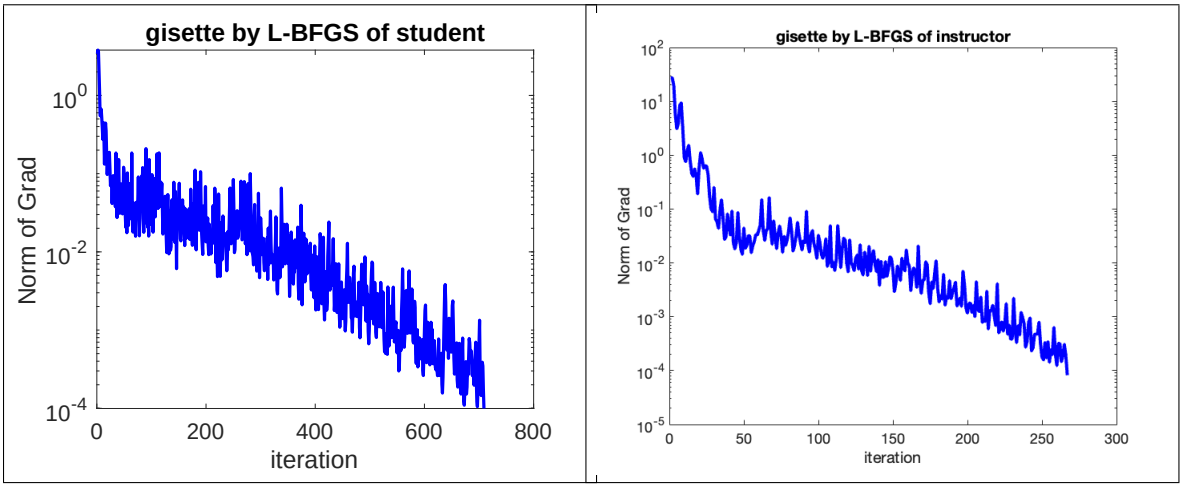
$$\left|\nabla f(x_k + \alpha_k d_k)^\top d_k\right| \leq c_2 \left|g_k^\top d_k\right|$$

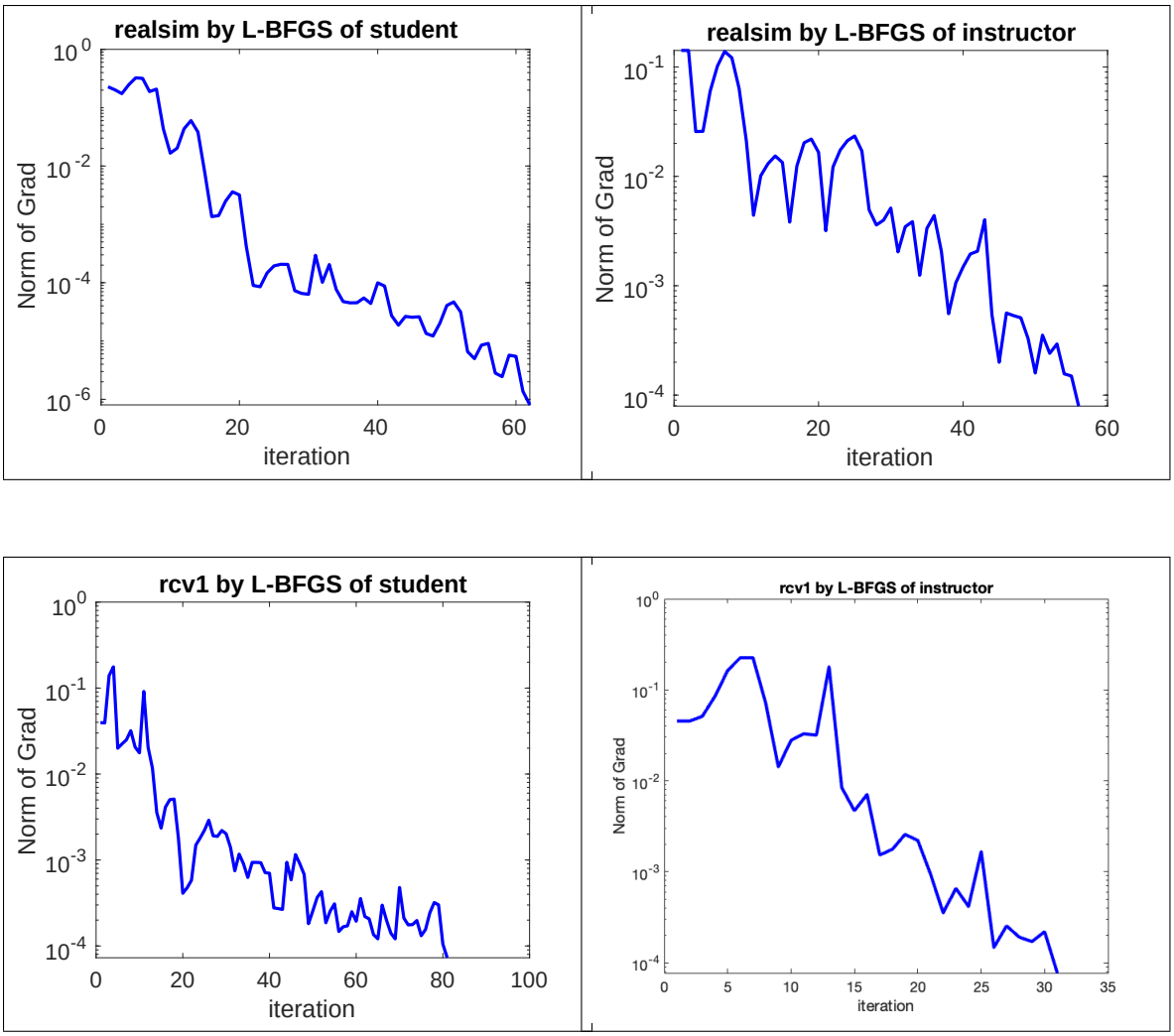
These conditions guarantee global convergence under standard assumptions such as convexity and Lipschitz continuity of the gradient.

2.3.2 Numerical Results and Observations of L-BFGS Algorithm

Dataset	λ_w	λ_b	**Role**	**Runtime (s)**	**Accuracy (%)**
gisette	1e-4	1e-4	Student	43.4404	97.80
			Reference	17.8191	98.10
rcv1	1e-4	1e-4	Student	1.8887	95.55
			Reference	1.0171	95.55
realsim	1e-5	0	Student	2.1986	83.62
			Reference	1.5809	97.25

Table 4: Comparison of student and Reference L-BFGS performance across datasets testing with different λ_w and λ_b values





Analysis of L-BFGS Results Across Datasets Our implementation of the L-BFGS method shows distinctive convergence behaviors across the three datasets:

- **Gisette Dataset:** Our L-BFGS implementation achieved 97.80% accuracy. The convergence plots reveal important differences: while both implementations show an initial steep decrease in gradient norm, our implementation exhibits a more oscillatory pattern with frequent small spikes throughout the optimization process. This suggests potential issues with our line search procedure or curvature estimation. Despite requiring more iterations (700 vs 300), our final model achieves nearly identical accuracy, demonstrating the robustness of L-BFGS for this challenging dataset.

- **Realsim Dataset:** Our implementation achieved an accuracy of 83.62%. The convergence plots offer a clear explanation: while both implementations show decreasing gradient norms over approximately 60 iterations, our implementation's curve shows larger, more irregular oscillations. These oscillations indicate potential issues with the Hessian approximation for this particular data distribution. The Reference's implementation demonstrates a more controlled descent pattern with periodic smaller oscillations that appear to be more effective at navigating the optimization landscape of this dataset.

- **RCV1 Dataset:** Our implementation acheived an accuracy of 95.55% on this dataset. The convergence plots show similar patterns of rapid initial descent followed by a more gradual decline, but with key differences in the fine structure. Our implementation shows a characteristic "staircase" pattern with alternating plateaus and sharp drops, particularly in the first 40 iterations. This pattern suggests that our algorithm might be taking larger steps followed by several smaller refinement steps. The Reference's implementation shows a somewhat smoother descent curve with fewer pronounced oscillations, which may explain its slightly faster runtime (1.02s vs 1.89s).

2.4 Conclusion

In this section, we explored two optimization methods for solving binary logistic regression problems: Accelerated Gradient Descent (AGD) and Limited-memory BFGS (L-BFGS). Our comparative analysis across three diverse datasets—Gisette, RCV1, and Realsim—revealed several important insights about these algorithms' performance characteristics.

The AGD method demonstrated strong performance with appropriate stepsize selection, especially on high-dimensional datasets like Gisette, where our implementation achieved 93.10% accuracy with faster runtime than the reference implementation. However, AGD's performance was characterized by oscillatory convergence patterns, particularly evident in the Realsim dataset. This behavior is a classic manifestation of momentum-based methods, where the algorithm occasionally "overshoots" the minimum before correcting its trajectory.

L-BFGS consistently outperformed AGD in terms of both convergence speed and accuracy, with our implementation achieving 97.80% accuracy on Gisette and identical 95.55% accuracy on RCV1 compared to the reference implementation. The quasi-Newton approach of L-BFGS leverages curvature information from previous iterations to accelerate convergence, which explains its superior performance. However, this advantage comes with increased computational complexity per iteration, as evidenced by the complex line search procedures

and Hessian approximation requirements.

L-BFGS typically required fewer iterations to converge, though with higher per-iteration computational cost than AGD, particularly evident on the high-dimensional Gisette dataset with its 5,000 features. Both algorithms demonstrated sensitivity to regularization parameters, but L-BFGS exhibited greater robustness to parameter selection, especially on the Realsim dataset. Notably, the sparse structure of RCV1 and Realsim datasets presented distinct convergence challenges compared to the dense Gisette dataset, underscoring how data characteristics fundamentally influence optimization dynamics.

In practical applications, the choice between AGD and L-BFGS involves a trade-off between implementation complexity and performance. AGD is simpler to implement and has well-understood theoretical guarantees with $O(1/k^2)$ convergence rate, making it suitable for large-scale problems where memory is a constraint. L-BFGS, while more complex, offers superior convergence properties and typically requires fewer iterations, making it more suitable for applications requiring high precision or when function evaluations are expensive.

Future work could explore adaptive variants of both algorithms, particularly adaptive momentum schemes for AGD and more sophisticated line search procedures for L-BFGS. Additionally, stochastic variants of these methods could be investigated for extremely large datasets where processing the full gradient becomes prohibitively expensive.

3 Contributions

R. Verma worked on the Binary Support Vector Machine problem and its solver. He also worked on the write up associated with this section of the report. Verma also worked on improving the overall flow of the report and the papers formatting.

Verma was aided by the lectures and codes provided by Professor Yangyang Xu to help develop the respective solver(s).