Minor Assignment 1 - CS771

Team Name: LossMinimizers (MLR-58)
Course: CS771 - Introduction to Machine Learning

Team Members
220023 AAYUSH SIDANA
220110 ALANKRIT GUPTA
220122 AMIPRIYA ANAND
220242 ASHUTOSH TRIPATHI
220465 ISHIKA BANSAL

Abstract

This report details the work for Minor Assignment 1 in CS771. We identified and corrected three coding errors in the provided SDCM algorithm implementation, achieving a test accuracy of 95.5%. We also conducted experiments to analyze the effects of various hyperparameters, including the regularization parameter C and the number of training points, on both test accuracy and the primal-dual objective convergence curves. Our findings are discussed in the sections below.

1 Problem 1.1 (Mess Made by Melbo)

This section addresses the tasks outlined in the assignment. All coding errors were confined to cell number 5 of the Jupyter notebook, as specified in the problem description.

1.1 Part 1: Discovering and Describing Errors

Task: *Discover all coding errors made by Melbo in implementing the SDCM algorithm. Describe the errors in your report along with a description of why you think it is an error.*

Based on a line–by–line comparison of the provided code and the C-SVM dual formulation, we identified the following errors in the SDCM implementation (cell #5):

1. **Incorrect projection onto the box constraints.** The dual variables must satisfy the box constraints $0 \leq \alpha_i \leq C$. The erroneous code projects with the condition `if newAlphai < C: newAlphai = C`, which forces almost all nonnegative updates to the *upper* bound C. *Correct behaviour:* project by clipping to $[0, C]$:

   ```
   if newAlphai > C: newAlphai = C;  if newAlphai < 0:  newAlphai = 0.
   ```

   This implements the proper projection required
2. **Wrong bookkeeping updates for w and b.** By definition $w = \sum_j \alpha_j y_j x_j$ and (with the bias treated as a dummy coordinate) $b = \sum_j \alpha_j y_j$. The proper updates should be:

$$w_{\text{SDCM}} = w_{\text{SDCM}} + (\text{newAlphai} - \alpha[i]) \cdot y[i] \cdot x$$
$$b_{\text{SDCM}} = b_{\text{SDCM}} + (\text{newAlphai} - \alpha[i]) \cdot y[i]$$

3. **Incorrect objective definitions** The primal being monitored should be

$$\mathcal{P}(w, b) = \tfrac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max\big(0,\, 1 - y_i(w^\top x_i + b)\big),$$

Preprint. Under review.

and the dual (in terms of the current w) should be

$$\mathcal{D}(\alpha) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \|w\|^2,$$

without any separate $\frac{1}{2}b^2$ term. The erroneous code adds $\frac{1}{2}b^2$ to both primal and dual, effectively regularizing the bias

After correcting the projection, the bookkeeping updates, and the objective definitions, the algorithm produces stable primal–dual convergence curves with a small duality gap and the test accuracy improves from $\sim 30\%$ to $\sim 95.5\%$, consistent with the expected behaviour of a properly implemented C-SVM with SDCM.

1.2 Part 2: Corrected Results

Task: *Once you have corrected Melbo's mistakes, what test accuracy do you get? Mention this and show what primal and dual objective curve do you get.*

Test Accuracy: 95.67%

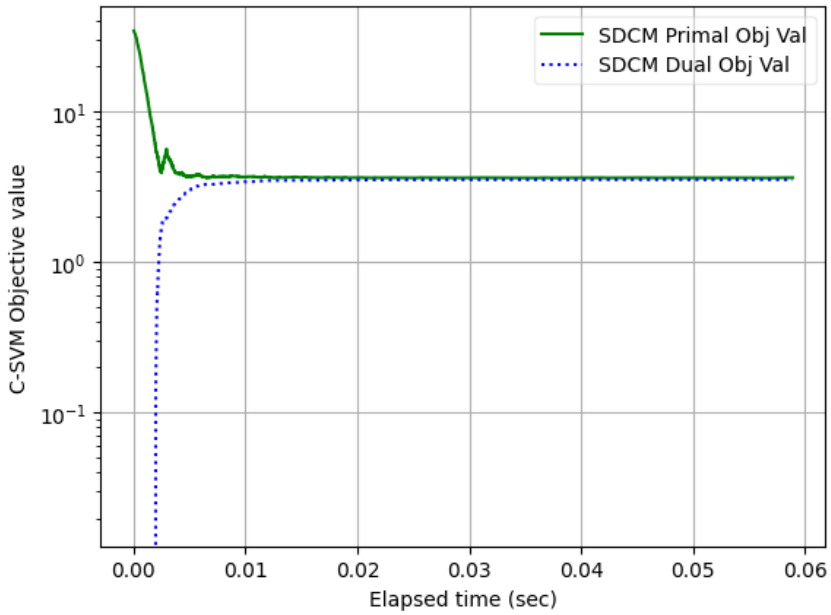


Figure 1: Primal and dual objective functions after correcting errors.

1.3 Part 3: Effect of Changing C

Task: *Change the value of C between very small values (e.g., 0.00001) and very large values (e.g., 100000). What changes do you observe in the test accuracy? What changes do you observe in the primal-dual objective curve?*

To study the effect of the C, we considered three representative cases: extremely low C, moderately sized C, and extremely high C. The behavior of both test accuracy and the primal-dual objective functions is summarized below.

Case 1: Extremely Low $C = 0.00001$

- **Test Accuracy:** $\approx 93\%$–94%
- **Explanation:** When C is very small, the hinge-loss term $C \sum \max(0, 1 - y_i(w^\top x_i))$ becomes negligible compared to the regularization term $\frac{1}{2}\|w\|^2$. The optimization therefore drives $w \to 0$, essentially ignoring misclassifications. This leads to an underfit classifier with modest accuracy.
- **Primal–Dual Curves:** Both the primal and dual objectives remain nearly flat. The primal objective is dominated by the regularization term, while the dual is restricted by $\alpha_i \in [0, C]$ with tiny C, producing minimal variation.

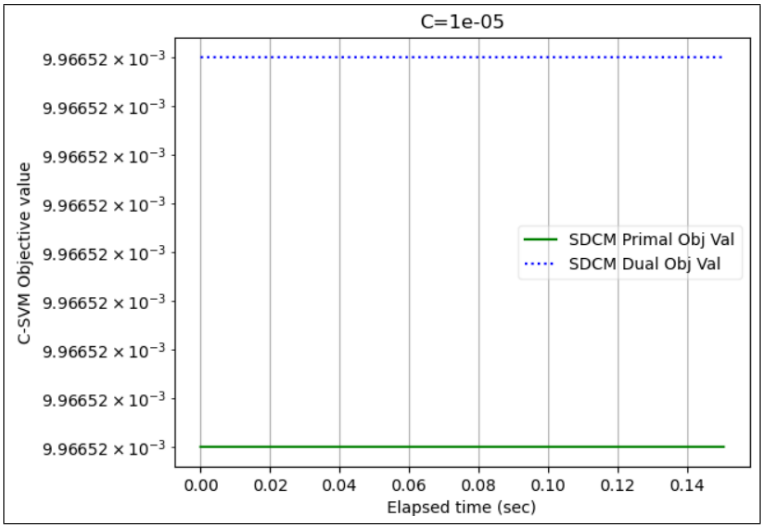


Figure 2: Primal and dual objective curves for extremely low $C = 0.00001$.

Case 2: Moderate $C = 0.01$

- **Test Accuracy:** Peaks around 95.5%–96%.
- **Explanation:** At moderate C, the balance between regularization and hinge loss is optimal. The model allows some misclassifications but maintains a sufficiently large margin, leading to the best generalization performance.
- **Primal–Dual Curves:** The primal (solid) and dual (dotted) curves converge rapidly and closely track each other, demonstrating strong duality with a vanishing duality gap.

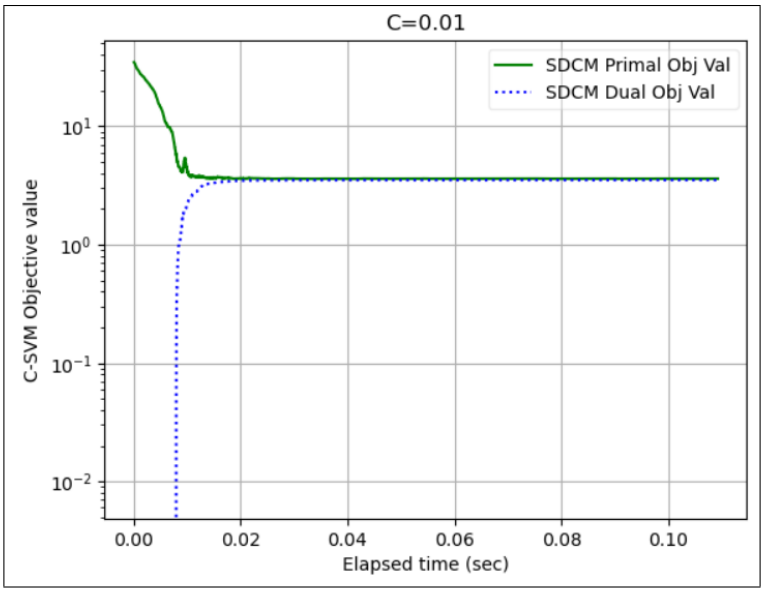


Figure 3: Primal and dual objective curves for moderate $C = 0.01$.

Case 3: Extremely High $C = 100000$

- **Test Accuracy:** Drops significantly to $\approx 77\%$.
- **Explanation:** A very large C forces the model to heavily penalize every hinge loss term, effectively trying to classify all training samples correctly. As a result, the learned classifier overfits and generalization accuracy deteriorates.
- **Primal–Dual Curves:** The primal objective initially explodes to values around 10^5 and then decays artificially, while the dual becomes negative and flattens near zero, hence was not rendered in graph

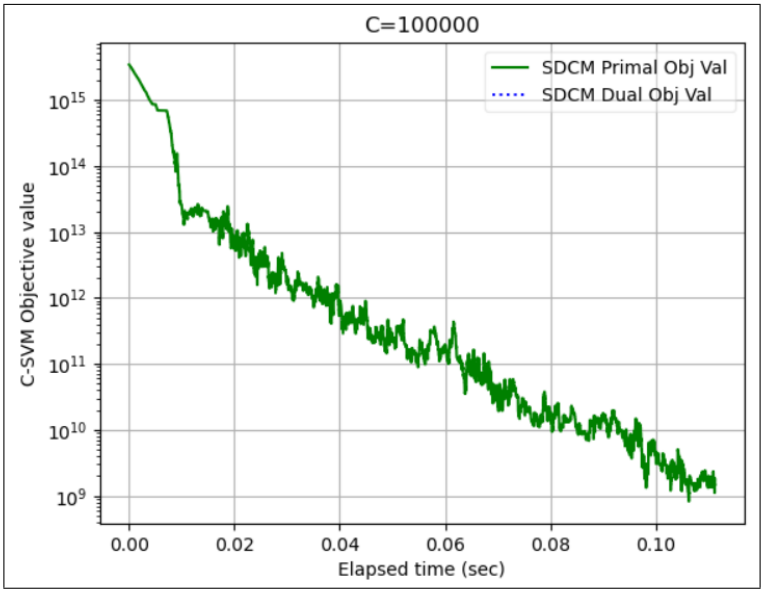


Figure 4: Primal and dual objective curves for extremely high $C = 100000$.

1.4 Part 4: Experimenting with Hyperparameters

Task: *Experiment with at least two of the hyperparameters mentioned below and comment on their effect on test accuracy and primal-dual convergence curve. You must report the test accuracies and the convergence curves in your report after changing these hyperparameters.*

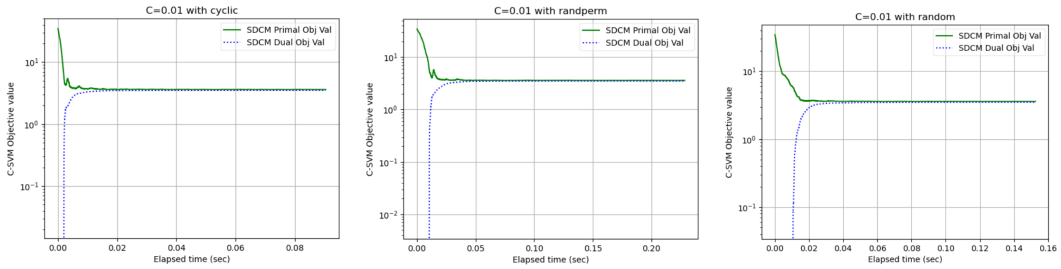
1.4.1 Part 4(a): Effect of changing the coordinate generator

Task: *Currently it is set to 'randperm'. What happens if we use 'cyclic' or 'random'?*

Test accuracy Representative results from our runs:

Generator	Accuracy
cyclic	**0.96067**
random	0.95784
randperm	0.95712

Summary: Across policies, accuracy differences are small ($\leq$0.3%). The generator primarily affects convergence speed and stability of the primal–dual curves, not final generalization.All policies converge to the same solution. *Random* seems bit jittery on zooming in. *Cyclic* seems the fastest.



(a) Cyclic (b) Randperm (c) Random

Figure 5: C-SVM objective vs. time (semilogy) for $C = 0.01$ under different coordinate generators.

1.4.2 Part 4(b): Effect of changing the initialization 'initDual'

Task: *The default initialization for the dual variables α is* all-C. *What happens if we instead initialize with all zeros or with random values in $[0, C]$?*

We compared three initialization strategies for α: all-C, all-zeros, and random uniform in $[0, C]$. The final test accuracies and convergence behaviours are summarized below.

Test Accuracy.

Initialization	Test Accuracy (%)
All-zeros	~ 95.9
All-C	~ 95.5
Random	~ 95.8

Observations.

- All three strategies converge to very similar accuracies (95.5–95.9%), consistent with the convexity of the SVM optimization problem.
- With **zero initialization**, the primal and dual curves start from the origin and steadily converge, yielding slightly higher accuracy than all-C.
- With **all-C initialization**, the primal begins at a larger value and settles more slowly, but converges to essentially the same optimum.
- With **random initialization**, convergence behaviour is intermediate: the curves fluctuate more initially but stabilize to nearly the same solution.

Conclusion. The choice of initialization has only a minor effect on final accuracy in SDCM ($\leq 0.5\%$ difference). However, it does influence the transient convergence path of the primal and dual objectives. This is expected since the underlying SVM objective is convex: regardless of initialization, the algorithm converges to (nearly) the same global optimum.

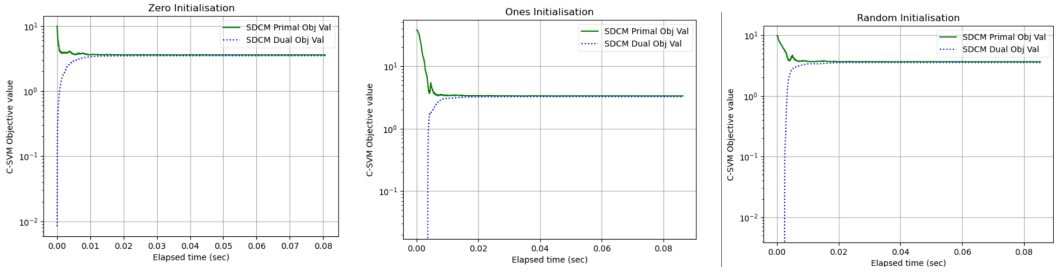


Figure 6: Effect of initialization on convergence: Zero initialization (left), All-C initialization (middle), Random initialization (right). All converge to nearly identical solutions with small differences in trajectory.

1.4.3 Part 4(c): Effect of Horizon (Number of Iterations)

Task: *Investigate the effect of changing the horizon (the total number of coordinate updates) in the SDCM algorithm. Does increasing it significantly yield a much better solution? Does decreasing it severely worsen the solution?*

We experimented with horizons ranging from 1000 to 100000 iterations and compared both the convergence of the primal–dual objective curves and the achieved test accuracy.

Test Accuracy.

Horizon	Test Accuracy (%)
1000	~ 85.0
10000	~ 95.0
20000	~ 95.5 (baseline)
100000	~ 95.79

Observations.

- With a **small horizon** (e.g., 1000 updates), the algorithm terminates too early. The dual objective has not yet caught up with the primal, leaving a large duality gap. As a result, the model is under-trained, and the test accuracy drops to around 85%.
- With a **moderate horizon** (e.g., 10000–20000), the primal and dual curves converge tightly and the duality gap nearly vanishes. This yields near-optimal test accuracies around 95%, demonstrating that this range is sufficient for stable convergence.
- With a **very large horizon** (e.g., 100000), the curves converge almost immediately and then remain flat for the remainder of the run. The accuracy improves only marginally (to $\sim 95.79\%$), showing that additional iterations provide negligible gains once convergence has been reached.

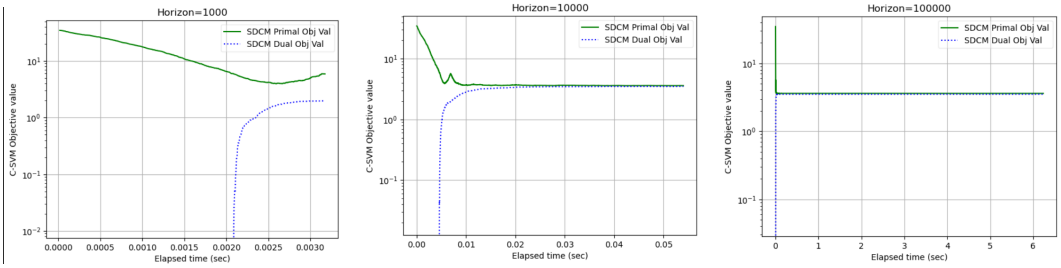


Figure 7: Effect of horizon on primal–dual convergence. Left: Horizon=1000 (large gap, premature stop). Middle: Horizon=10000 (converged). Right: Horizon=100000 (identical convergence, no further gains).

Conclusion. Increasing the horizon beyond 20000 does *not* significantly improve the solution, as convergence has already been reached. However, decreasing it too much (e.g., to 1000) severely worsens the solution due to premature termination. Thus, an intermediate horizon of 10000–20000 provides an optimal trade-off between training time and accuracy.

1.4.4 Part 4(d): Effect of Number of Training Points

Task: *Investigate whether increasing or decreasing the number of training points substantially affects test accuracy and the primal–dual convergence behaviour.*

We varied the number of training points from 100 to 10000, keeping all other hyperparameters fixed, and measured both test accuracy and the primal–dual convergence curves.

Test Accuracy.

Training Points	Test Accuracy (%)
100	~ 83.4
1000	~ 96.1
10000	~ 97.4

Observations.

- With only **100 training points**, the classifier underfits: accuracy drops to $\sim 83\%$. The primal and dual curves converge quickly but to a suboptimal solution, since the model has insufficient data to learn a good decision boundary.
- At the baseline of **1000 points**, accuracy rises sharply to $\sim 96\%$, and the primal and dual objectives converge smoothly with a small duality gap, showing that the model has enough samples to generalize well.
- With a very large dataset of **10000 points**, accuracy improves marginally further ($\sim 97.4\%$). However, convergence is noticeably slower: the primal objective decreases gradually and the dual catches up only after longer elapsed time. This suggests that larger datasets require a longer horizon (more than 20000) to fully close the duality gap.

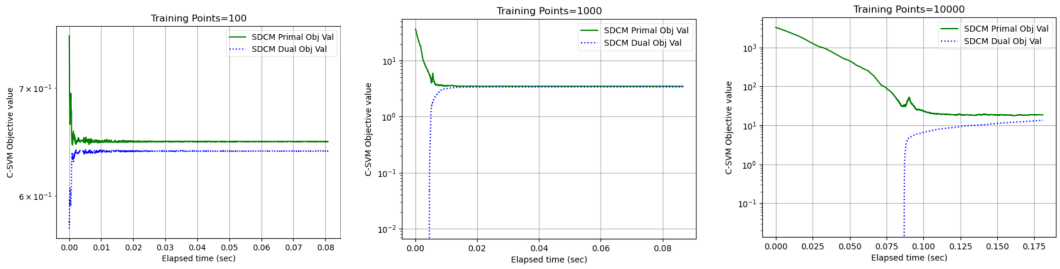


Figure 8: Effect of training points on convergence. Left: 100 points (fast convergence, low accuracy). Middle: 1000 points (good convergence and accuracy). Right: 10000 points (slow convergence, highest accuracy).

Conclusion. Increasing the training set size improves accuracy, but with diminishing returns (from 96% at 1000 to only 97.4% at 10000). Convergence also becomes slower for large datasets, often requiring an increased horizon to allow the duality gap to vanish. Conversely, with very few training points, convergence is fast but accuracy suffers significantly due to underfitting.

1.4.5 Part 4(e): Random chance

Task: *Even with all hyperparameters and settings fixed, does re-running the notebook from start to end change the test accuracy? If yes, what is the reason?*

Observation. When executing the entire Jupyter notebook multiple times with identical hyperparameters (e.g., $C = 0.01$, horizon = 20000, coordinate generator = `randperm`), the final test accuracy is not identical across runs. We observed variations on the order of 0.5–1% (e.g., between 95.3% and 96.0%).

Explanation. The source of variability is the inherent *stochasticity* of the SDCM algorithm:

- The `random` or `randperm` coordinate selection schemes sample coordinates in a random order. Even though the expectation of the updates is unbiased, different random sequences of coordinates lead to slightly different intermediate iterates.
- Since SDCM is not a deterministic solver but a stochastic one, the optimization path is non-unique. However, due to convexity of the SVM objective, all runs converge near the same optimum, so the variation in test accuracy remains small.

Conclusion. Yes, running the notebook repeatedly can yield slightly different test accuracies, even with the same hyperparameters. This is due to the randomization in coordinate selection and initialization. The differences are minor because the underlying problem is convex and the stochastic solver still converges close to the global optimum, but the exact trajectory of updates varies across runs.