Minor Assignment 1

Vihaan Sapra
231149
svihaan23@iitk.ac.in

Nischay Agarwal
230705
nischaya23@iitk.ac.in

Aryamann Srivastava
230211
aryamanns23@iitk.ac.in

Dharajiya Yug Harshadbhai
230362
dharajiya23@iitk.ac.in

Jain Shlok Nilesh
230493
jainshlok23@iitk.ac.in

Shravan Agrawal
230984
ashravan23@iitk.ac.in

Abstract

This report is submitted as Minor Assignment 1 for the course CS 771, by the Group **MLVortex**.

1 Finding errors in code

1.1 Error 1 - Incorrect clipping of dual variable

Incorrect Line of Code:

```
if newAlphai < C:
    newAlphai = C
if newAlphai < 0:
    newAlphai = 0
```

Why error: This forces *all* values below C up to C, never clipping values above C, and can even map negative values to C because newAlphai(initially if less than 0) will be C so 2nd condition won't meet.

Fix:

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

Alpha needs to be constrained between 0 and C, previously it was constrained from C to positive infinity

$$0 \leq \alpha_i \leq C,$$

1.2 Error 2 - Wrong update of the primal weights

Erroneous code:

```
w_SDCM = w_SDCM - (newAlphai + alpha[i]) * y[i] * x
b_SDCM = b_SDCM - (newAlphai + alpha[i]) * y[i]
```

Preprint. Under review.

Why error: Let $\Delta = \alpha_i^{\text{new}} - \alpha_i$. Since $w = \sum_j \alpha_j y_j x_j$ and (with regularized bias) $b = \sum_j \alpha_j y_j$, a single-coordinate change must update as

$$w \leftarrow w + \Delta\, y_i x_i, \qquad b \leftarrow b + \Delta\, y_i.$$

The incorrect code uses a minus sign and (new + old) instead of the required difference.
Fix:

```
delta = (newAlphai - alpha[i]) * y[i]
w_SDCM = w_SDCM + delta * x
b_SDCM = b_SDCM + delta
```

1.3 Error 3-Off by one in getRandpermCoord

Incorrect behavior: With state $(\text{idx}, \text{perm})$, the code increments the index before reading, so the very first element perm$[0]$ of the first epoch is skipped.

Why error: This creates a biased coordinate visitation (first epoch never visits the first element).

Fix (read-then-advance):

```
def getRandpermCoord(state):
    idx, perm = state
    d = len(perm)
    curr = perm[idx]            # read current
    idx += 1                    # advance for next time
    if idx >= d:                # wrap and reshuffle at epoch boundary
        idx = 0
        perm = np.random.permutation(d)
    return (curr, (idx, perm))
```

1.4 Error 4- Missing a term in the 1-D optimum

Incorrect code:

```
newAlphai =  (1 - y[i]*(x.dot(w_SDCM) + b_SDCM)) / normSq[i]
```

Why error: In dual coordinate ascent for CSVM (with bias treated as a regularized feature), the unconstrained update is

$$\alpha_i^{\text{new}} \;=\; \alpha_i \;+\; \frac{1 - y_i\,(w^\top x_i + b)}{\|x_i\|_{\text{eff}}^2},$$

where $\|x_i\|_{\text{eff}}^2 = \|x_i\|^2$ or $\|x_i\|^2 + 1$ depending on whether b is regularized as an extra feature. If we don't take α_i it will go to the wrong point.

Fix:

```
newAlphai = alpha[i] + (1 - y[i]*(x.dot(w_SDCM)+b_SDCM))/normSq[i]
```

2 Test Accuracy after Running

After removing all the bugs form the code, the new accuracy that we are getting is **95.72%**.
The primal and dual objective curve is shown in Figure 1.

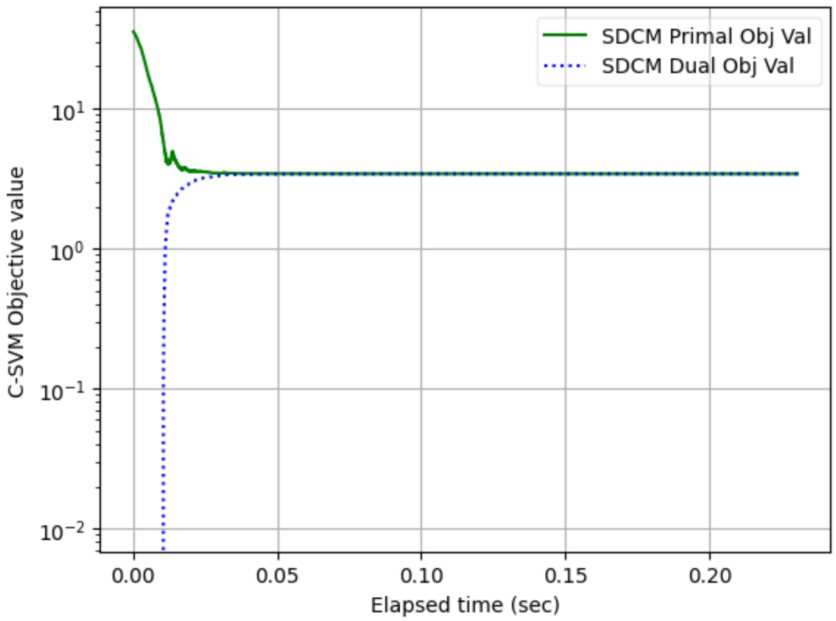

Figure 1: Primal Dual objective curve for $C = 0.01$

3 Effect of Regularization Parameter C

We next studied the impact of varying the SVM regularization parameter C, which controls the trade-off between maximizing the margin and minimizing hinge loss penalties.

3.1 Experimental Setup

We trained the CSVM model with three different values of C: a small value ($C = 0.00001$), a moderate value ($C = 0.01$), and a large value ($C = 10000$). All other hyperparameters, including coordinate generator policy and initialization, were kept fixed (`randperm` and zero initialization). We then compared the test accuracy and convergence behavior of the primal–dual objectives.

3.2 Results

C Value	Test Accuracy
0.00001	0.9284
0.01	0.9572
10000	0.9629

Table 1: Test accuracy of CSVM under different values of C.

Observation:

- With a small $C = 0.00001$, the model is strongly regularized. This yields smoother primal–dual curves but underfits, leading to lower accuracy.
- With a moderate $C = 0.01$, the model achieves the best balance between margin maximization and classification accuracy.
- With a large $C = 10000$, the model tries to perfectly fit the training data. Accuracy improves slightly, but primal–dual curves show slower convergence and the risk of overfitting increases.

3.3 Primal–Dual Objective Curves

The following plots show the primal and dual objective values as functions of runtime for different values of C.

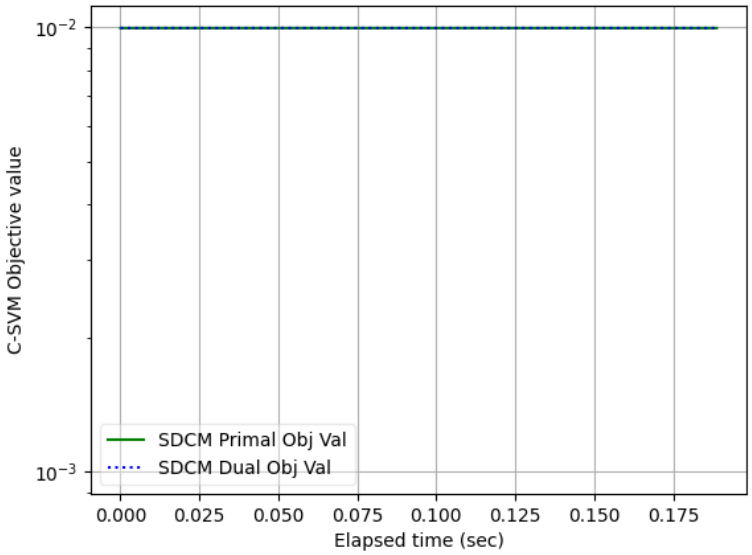

Figure 2: Primal–Dual objective curves for $C = 0.00001(10^-5)$ (strong regularization).

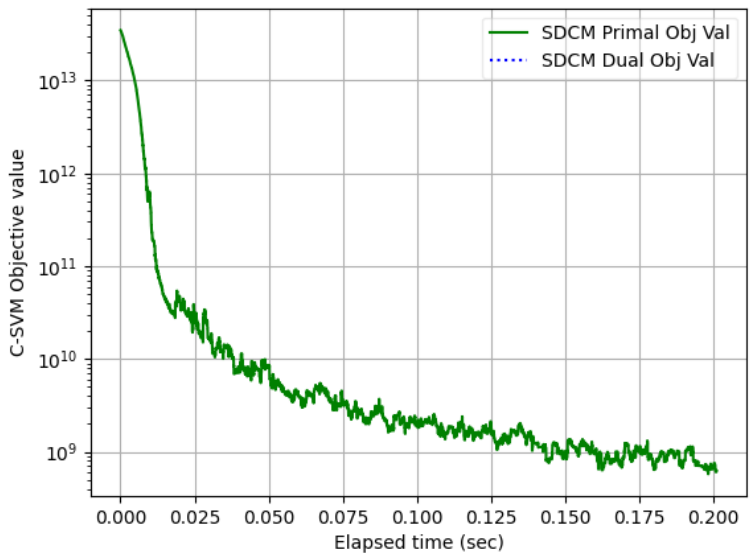

Figure 3: Primal–Dual objective curves for $C = 10000(10^4)$ (weak regularization).

4 Effect of Hyperparameters

We experimented with two hyperparameters: running the code again with the same parameters to check the random chance and the initialization of dual variables (zeros, random). For each setting we trained the SDCM solver under identical conditions and recorded test accuracy and primal-dual convergence curves.

4.1 (b) Initialization of Dual Variables

We compared initializing α to all zeros vs random values, while keeping the coordinate generator fixed to `randperm`.

Initialization	Test Accuracy
All-Zero	0.9567
Random	0.9569
All-C	0.9572

Table 2: Test accuracy under different initializations of dual variables.

Observation: All initializations converged to essentially the same final accuracy.

1. **Zero initialization:** Starting from $\alpha = 0$ represents a hard-margin baseline and yields a larger initial primal–dual gap, so more coordinate updates are needed, causing slower early convergence.
2. **All-C initialization:** Setting all duals to C acts like a warm start biased toward the penalty-saturated region. This reduces the initial primal–dual gap when many optimal α's lie near the bound, leading to slightly faster early convergence, though the final accuracy is unchanged.
3. **Random initialization:** Starting with small random values for α provided a mild warm start, lowering the initial gap and giving faster early convergence than zero initialization, but with no long-run effect on accuracy.

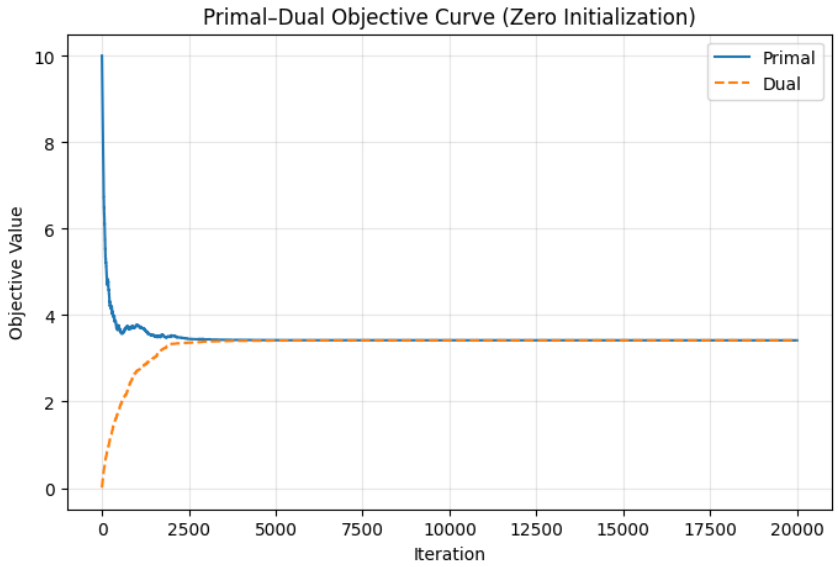

Figure 4: Primal–Dual objective curve for zero initialization of dual variables.

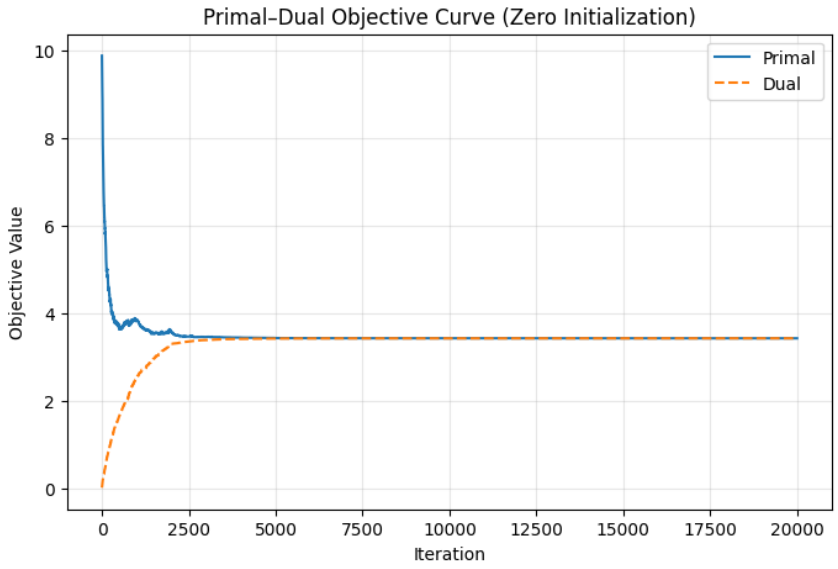

Figure 5: Primal–Dual objective curve for all C initialization of dual variables.

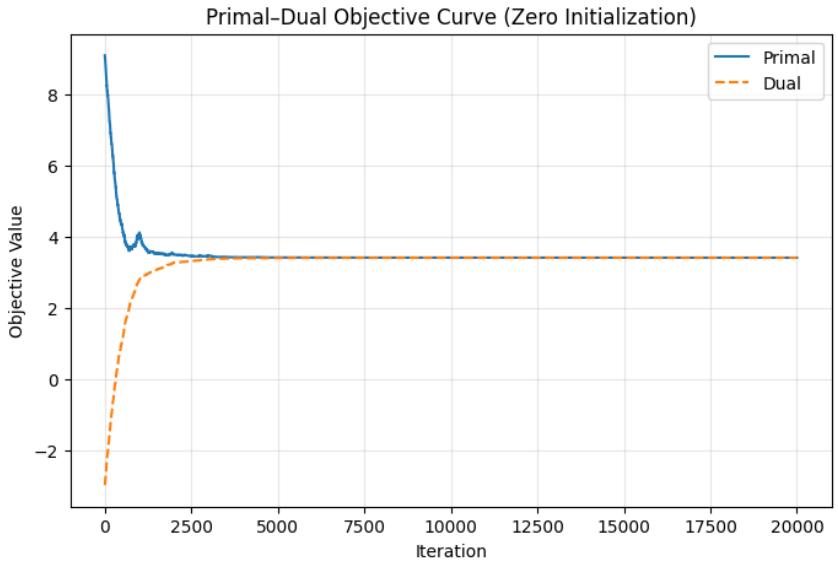

Figure 6: Primal–Dual objective curve for random initialization of dual variables.

4.2 (e) Randomness

Yes, test accuracy can change between runs even with the same settings. This happens because the code uses **randomness** in multiple places:

- **Random permutations:** NumPy functions like `np.random.permutation(d)` generate random permutations of length d. Unless we fix a random seed, each run gives a different ordering of the data.
- **Train-test split:** The function `train_test_split` in `scikit-learn` randomly divides the dataset into training and testing sets. Without fixing `random_state`, the split changes on each run, leading to different accuracies.

As a result, the objective function curves and final test accuracies vary slightly between runs, even when all hyperparameters remain unchanged.

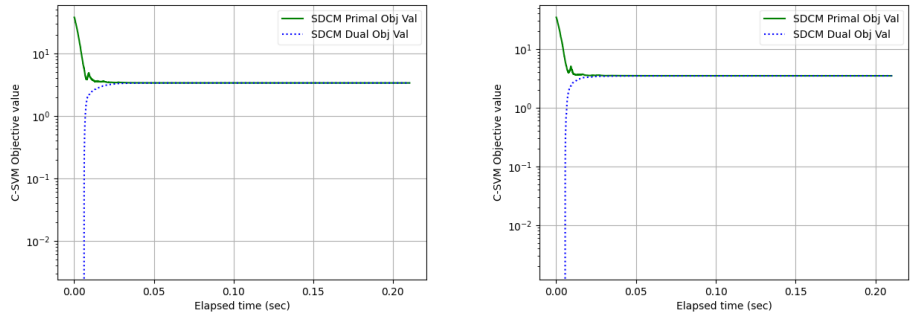

Figure 7: Two runs of the same experiment showing different objective function curves due to randomness.

Run	Test Accuracy
1	0.9538585858585859
2	0.9525656565656566

Table 3: Variation in test accuracy across two runs with identical hyperparameters.

Yes, test accuracy change between runs even with the same settings. This is because code uses randomness in generating a random permutation for coordinates. NumPy (np.random.permutation(d)) generates random permutation of length d, so unless we fix a random seed, each run gives slightly different results.

References

1. **GeeksForGeeks.** *Support Vector Machines (SVM) Algorithm*. Retrieved from GeeksForGeeks
2. **Python Documentation.** *Standard Libraries, Built-in Functions*. Retrieved from Python Docs
3. **Lecture Notes.** *By Prof. Purushottam Kar*. Retrieved from ML Course Page