# CS771 Minor Assignment 1

**MLR-48 : Gradient Gang**

## 1 Question-1

The following are all the errors found in cell 5:

1. **Line 11**: New optimal value of alpha_i. Gradient-based update should incorporate the older / previous values of alpha_i as well.

$$\alpha_i^{new} = \alpha_i^{old} + \frac{1 - y_i\big(w \cdot x_i + b\big)}{\|x_i\|^2}$$

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

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

2. **Line 14**: Making sure constraints are satisfied. Alpha has to be bounded in [0, C]

```
# Original
if newAlphai < C:
        newAlphai = C
```

```
# Modified
if newAlphai > C:
        newAlphai = C
```

3. **Line 21-22**: Updating the model vector and bias values should add the delta instead of subtracting it as per Stochastic Dual Coordinate Ascent [2]. The updated formulae are

$$\Delta\alpha_i = \alpha_i^{new} - \alpha_i^{old}$$

$$w \leftarrow w + \Delta\alpha_i \, y_i \, x_i$$

$$b \leftarrow b + \Delta\alpha_i \, y_i$$

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

```
# Modified
delta = newAlphai - alpha[i]
w_SDCM = w_SDCM + delta * y[i] * x
b_SDCM = b_SDCM + delta * y[i]
```

4. **Line 32-33**: Primal and dual objective functions. As per the equations mentioned in (1), these functions have been updated.

(a) Primal objective function

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max\left(0,\, 1 - y_i(w \cdot x_i + b)\right)$$

(b) Dual objective function

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle$$

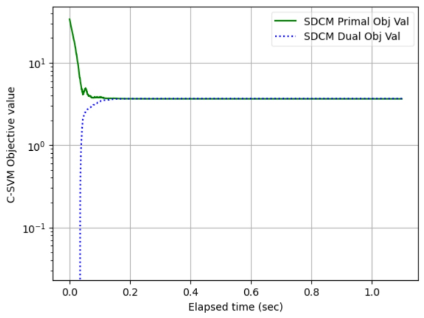
$$subjectto \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

```
# Original
objPrimal = 0.5 * w_SDCM.dot( w_SDCM ) + 0.5 * b_SDCM * b_SDCM
    + C * np.sum(hingeLoss)
objDual = np.sum( alpha ) - 0.5 * np.square(np.linalg.norm(
    w_SDCM)) - 0.5 * b_SDCM * b_SDCM
```

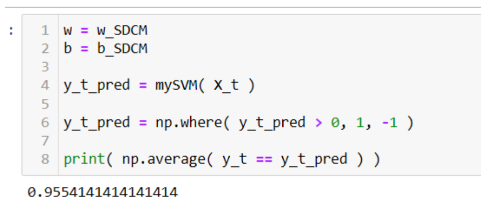
```
# Modified
objPrimal = 0.5 * w_SDCM.dot( w_SDCM ) + C * np.sum(hingeLoss)
term = alpha * y
objDual = np.sum(alpha) - 0.5 * term @ (X @ X.T @ term)
```

## 2 Question-2

Accuracy obtained: 95.54 %



(a) Plot of Primal and Dual objective functions



```
w = w_SDCM
b = b_SDCM

y_t_pred = mySVM( X_t )

y_t_pred = np.where( y_t_pred > 0, 1, -1 )

print( np.average( y_t == y_t_pred ) )
```

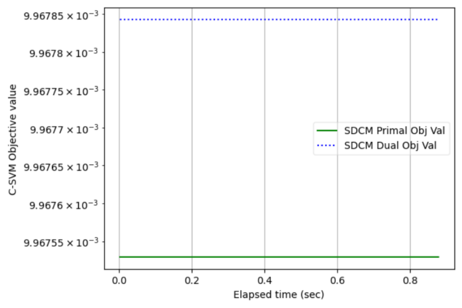
```
0.9554141414141414
```

(b) Test Accuracy

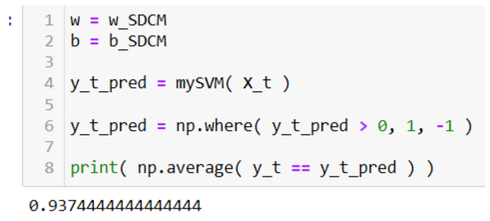
## 3 Question-3

1. **C = 0.00001**

   For very small values of C, the model will allow many mis-classifications. Since alpha will be confined to [0, C], w and b will converge to near-zero values, and the primal and dual objectives will also stabilize to near-zero values. But the test accuracy remains largely unchanged. Accuracy obtained: 93.7 %



(a) Plot of Primal and Dual objective functions

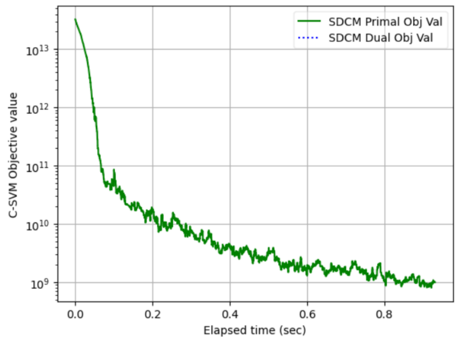


```
w = w_SDCM
b = b_SDCM

y_t_pred = mySVM( X_t )

y_t_pred = np.where( y_t_pred > 0, 1, -1 )

print( np.average( y_t == y_t_pred ) )
```

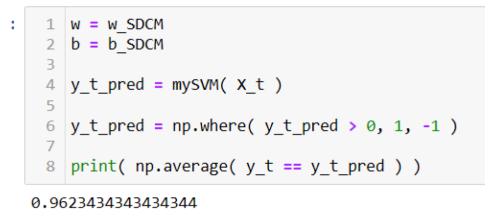
```
0.9374444444444444
```

(b) Test Accuracy

2. **C = 10000**
   For large values of C, the model essentially becomes a hard-margin classifier and tries to classify all training points correctly. The plots of primal and dual objectives do not seem to converge because the gradient might be too large.
   But the test accuracy has increased to : 96.23 %



(a) Plot of Primal and Dual objective functions



```
w = w_SDCM
b = b_SDCM

y_t_pred = mySVM( X_t )

y_t_pred = np.where( y_t_pred > 0, 1, -1 )

print( np.average( y_t == y_t_pred ) )
```

```
0.9623434343434344
```

(b) Test Accuracy

## 4 Question 4

We experimented with the following hyperparameters:

a) **Changing the coordinate generator**:
   Tested with *cyclic, random* & *randperm* coordinate generators. Here are some findings -
   1. *cyclic* converges fast, because it does not have to deal with the overhead of generating random numbers/permutations. The slight oscillation shown by cyclic implies non-optimal convergence due to correlation between features as seen in fig.4.

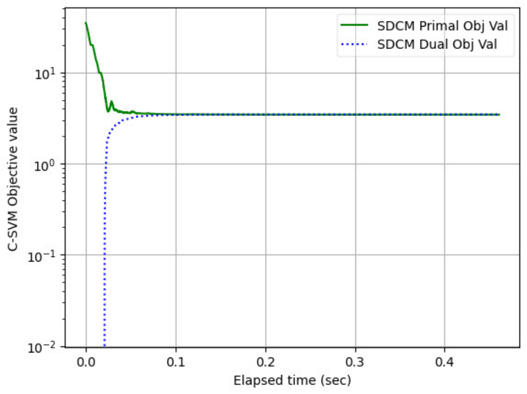


Figure 4: Plot of primal and dual with cyclic coordinate generator

2. *random* has a smooth convergence, but due to redundant updates (selecting same few coordinates repeatedly), it takes slightly longer to converge and has some fluctuations as seen in fig.5.

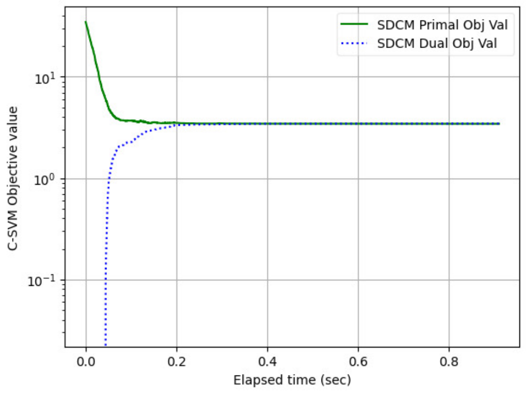


Figure 5: Plot of primal and dual with random coordinate generator

3. *randperm* is the best performing overall because it is more robust against feature correlations due to randomization. It also does not waste as many computations as random, on redundant updates as seen in fig.6.

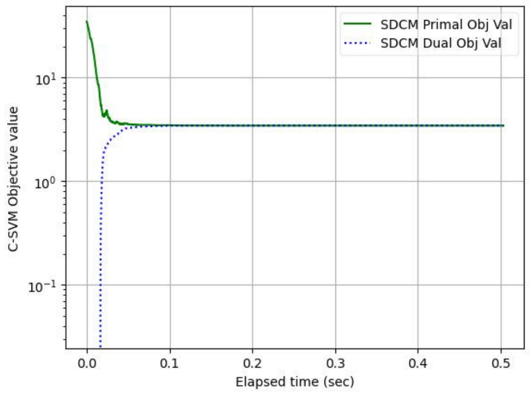


Figure 6: Plot of primal and dual with randperm coordinate generator

b) **Changing the initialization initDual**:

Since C-SVM optimization is a convex function, it will have a single global optimum. Changing the initialization will have no considerable effect on the plots since coordinate descent will converge on the same solution. The accuracy of the model is dependent on the support vectors, and random initialization does not determine which points become the support vectors. Hence, even accuracy remains largely unchanged.

c) **Changing the horizon (number of iterations)**:

The number of iterations has a direct affect on the optimum obtained. We must choose sufficient iterations so that the global optimum is reached and the training time is not too high.

- Too few iterations:

  If we stop early, we may get a sub-optimum and therefore the margin is not correct, leading to higher hinge loss. From Figure 7 we observe the primal and dual objectives have not yet converged. This also causes the model to have low test accuracy (65.6 %).

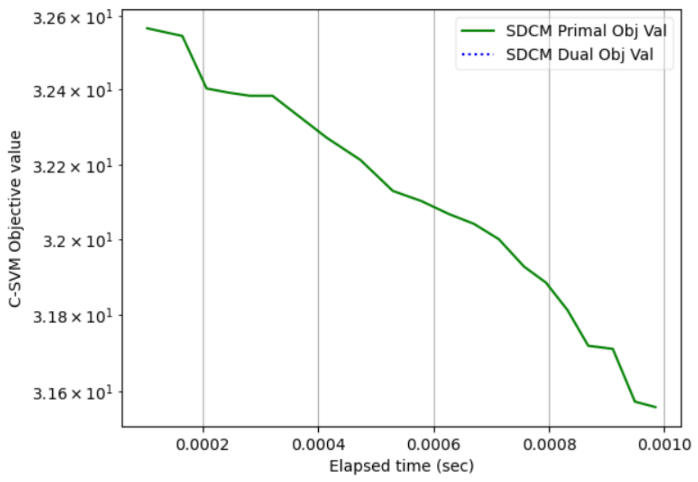


Figure 7: Plot for horizon=20

- Too many iterations:
  As the iterations continue, the updates become smaller corrections. The primal and dual objectives converge and the accuracy improves. However, a very high number of iterations unnecessarily increases training time.

d) **Changing the number of Training points**:
The number of training points has a big impact on the objective plots and test accuracy.

- Few training points:
  With too few training points we may get few support vectors and hence the hyperplane will not be stable depending on the initialization and C. The primal and dual objectives may converge quickly but can fluctuate. Test accuracy drops significantly (61.5 %)

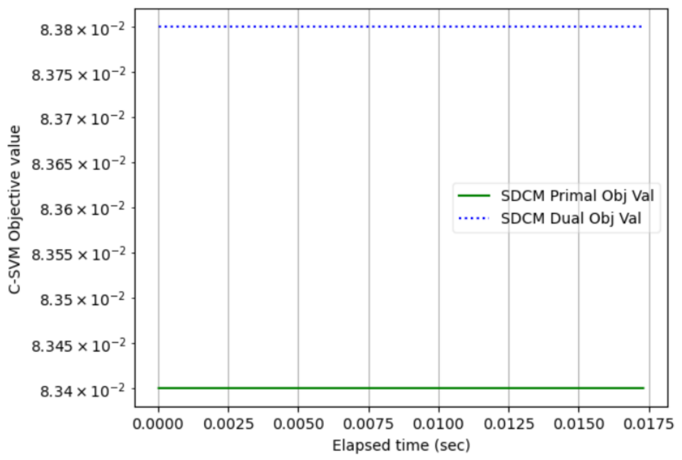


Figure 8: Plot with Train size=10

- Too many training points:
  With more training points, the hyperplane is more robust, the accuracy stabilizes, but we may end up overfitting the model.

e) **Running the notebook multiple times**:
Since all hyperparameters such as C, horizons, and the number of training points are kept constant, the resulting plots and test accuracy remain largely unchanged. Minor fluctuations in test accuracy may still occur due to the inherent randomness in the train-test split process, but these variations are not significant.

## References

[1] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2nd edition, 2018.

[2] Shai Shalev-Shwartz and Tong Zhang. *Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization.* Journal of Machine Learning Research, 2013.