CS771: Assignment 1

7 September 2019

Abhyuday Pandey	Ayush Kumar	Deepesh Lall	Srinjay Kumar	Umang Malik
170039	170195	170234	170722	170765

Solutions

1. The Lagrangian for the dual problem after introducing the dual variable $\boldsymbol{\alpha}$ is given by :

$$\boldsymbol{\mathcal{L}}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||_2^2 + C \cdot ||\boldsymbol{\xi}||_2^2 - \langle \boldsymbol{\alpha}, \boldsymbol{\xi} \rangle + \sum_{i=1}^{n} \alpha_i (1 - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle)$$

2. On applying first order optimality, i.e. setting gradients to 0, we get :

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{w}} = 0 \implies \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x^i} = 0 \implies \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x^i}$$

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\xi}} = 0 \implies 2C\boldsymbol{\xi} - \boldsymbol{\alpha} = 0 \implies \boldsymbol{\xi} = \frac{\boldsymbol{\alpha}}{2C}$$

Substituting these values in the Lagrangian, the dual problem (D2) becomes :

$$\implies \underset{\alpha_i \geq 0}{\boldsymbol{argmax}} \left(\frac{1}{2} \left\langle \sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x^i}, \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j} \right\rangle + \frac{C}{4C^2} \langle \boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle - \frac{1}{2C} \langle \boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle + \sum_{i=1}^{n} \alpha_i \left(1 - y^i \left\langle \sum_{j=1}^{n} \alpha_j y^j \boldsymbol{x^j}, \boldsymbol{x^i} \right\rangle \right) \right)$$

$$\implies \underset{\alpha_i \geq 0}{\boldsymbol{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \frac{1}{4C} ||\boldsymbol{\alpha}||_2^2 \right)$$

3. b. Used Coordinate descent on P1 to arrive at the optimal value.

$$P1 : \underset{\boldsymbol{w} \in \mathbb{R}^d}{\boldsymbol{argmin}} \ \frac{1}{2}||\boldsymbol{w}||_2^2 + C \cdot \sum_{i=1}^{n} ([1 - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle]_+)^2 \qquad (P1)$$

Let $f(\boldsymbol{w})$ be the above objective function. Since we only want $(\nabla f)_t$ (t^{th} coordinate)

$$\frac{\partial f}{\partial w_t} = w_t + C \cdot 2 \cdot \sum_{i=1}^{n} ([1 - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle]_+ \cdot y^i \cdot \nabla l_{hinge}(y^i \left\langle \boldsymbol{w}, x^i \right\rangle, x_t^i))$$

here, $\nabla l_{hinge}(\boldsymbol{s}, \boldsymbol{x}) = \begin{cases} -\boldsymbol{s}, & \text{if } \boldsymbol{s} \cdot \boldsymbol{x} < 1 \\ 0, & \text{otherwise} \end{cases}$

In each iteration, we cyclically choose a coordinate (let j) from a randomly generated permutation (generating once a cycle is completed) and update $\boldsymbol{w}$ as :

$$w_i^{\text{new}} = \begin{cases} w_i, & \text{if } i \neq j \\ w_i - (\frac{\partial f}{\partial \boldsymbol{w}})_t * \eta_t, & i = j \end{cases}$$

The choice of cyclically choosing coordinates from random permutation was decided after comparing it with other methods like cyclic coordinate and random coordinate.

Here, the value of Δx_t is chosen by using the AdaDelta algorithm, which uses a moving window of the previous gradient updates. This way, it uses the previous values of gradient continues learning even when many updates have been done.

$$\Delta x_t = -\frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$$

Here, RMS means root mean square of the values.

e. Used Stochastic Dual Coordinate Ascent on (D2) to arrive at the optimal value of $\boldsymbol{\alpha}$. Sampled the coordinate i to be updated cyclically using ***getCyclicCoord***().

$$\text{Let, } f(\boldsymbol{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \frac{1}{4C} ||\boldsymbol{\alpha}||_2^2 \right)$$

$$\text{Then, } \frac{\partial f}{\partial \alpha_i} = 1 - \alpha_i ||\boldsymbol{x^i}||_2^2 - y^i \sum_{j \neq i} \alpha_j y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \frac{\alpha_i}{2C} \qquad \text{(using } (y^i)^2 = 1) \ \ ...(\mathbf{1})$$

The summation term in the above expression is an *O(nd)* calculation while the other terms are *O(d)* or less. To reduce this overhead, we modify the expression a bit. We substitute,

$$y^i \sum_{j \neq i} \alpha_j y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle = y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle - \alpha_i ||\boldsymbol{x^i}||_2^2 \qquad \text{where, } \boldsymbol{w} = \textstyle\sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x^i} \ ..(\mathbf{2})$$

$$\implies \frac{\partial f}{\partial \alpha_i} = 1 - \frac{\alpha_i}{2C} - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle$$

When we update α_i, we also need to modify $\boldsymbol{w}$ accordingly.

$$\alpha_i' = \alpha_i + s \cdot \frac{\partial f}{\partial \alpha_i}$$

$$\boldsymbol{w} = \boldsymbol{w} + (\alpha_i' - \alpha_i) y^i \boldsymbol{x^i}$$

where, α_i' is the new value of α_i and s is the step length. Using an initial step length of $\eta_0 = 1.0$ and the step function $\eta = \frac{\eta_0}{\sqrt{t}}$ and initializing $\boldsymbol{\alpha}$ with a zero vector, and $\boldsymbol{w}$ with $\sum_{i=1} \alpha_i^n y^i \boldsymbol{x^i}$ i.e. again a zero vector, we ran the SDCA algorithm.

f. Used Stochastic Coordinate Maximization on (D2) to arrive at the optimal value of $\boldsymbol{\alpha}$. To get a an extrema of D2 we equate (**1**) to zero.

$$\alpha_i = \frac{1 - y^i \sum_{j \neq i} \alpha_j y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle}{\frac{1}{2C} + ||\boldsymbol{x^i}||_2^2}$$

Using (**2**) to simplify the expression, we get

$$\alpha_i = \frac{1 - y^i \boldsymbol{w^T} \boldsymbol{x^i} + \alpha_i ||\boldsymbol{x^i}||_2^2}{\frac{1}{2C} + ||\boldsymbol{x^i}||_2^2}$$

We sampled the coordinate i, and picked a uniformly random i from $i \in [1, n]$ using **rnd.randint** in Python.

4. b. Coordinate descent on P1:
We tried various different step functions. Comparison between any two different step functions was done by using 2-fold validation.

- First, we tried constant step function. For small values of η (around 0.00005, the descent was very slow. For larger values (0.001), the objection function started to explode.

- Next, we used the function $\eta/\sqrt{t}$, after checking for $\eta = [0.001, 0.0015, 0.002, 0.005, 0.01]$, we found $\eta = 0.0015$ worked best. The solver behaved very well for C=1 and T=30 seconds, achieving an optimum value of around 3600 on a held out validation set of size 14,000 data points.
- Next, to make use of the gradient changes, we conducted a literature survey on the internet and decided to try two more step functions, namely AdaGrad[1] and AdaDelta [2].
- In case of AdaGrad, the gradient was very high initially, due to which the step length was drastically reduced. This resulted in very slow convergence.

Next, to analyze the effect of C and to compare the time of convergence among the choices, we decided to use $C = [0.1, 0.6, 1.1, ..., 5]$. For each C we ran the solver with $[10, 20, 30]$ seconds as timeout.

Next, we implemented AdaDelta and compared it with $\eta/\sqrt{t}$ using the mentioned method. We observed the following:

- For C around 1-2.5, AdaDelta performed slightly better (or equal) compared to sqrt-t.
- For C outside the range, AdaDelta performed significantly better. In fact, for large values of C, sqrt-t resulted in very high value of Objective function.

Since AdaDelta won 28:2 against sqrt-t, we decided to use it.

Hyperparameter tuning: ρ in AdaDelta represents the weight of the previous values taken by the gradient. Using 2-fold validation, we tried for $\rho = [0.90, 0.95, 0.98]$ and found that 0.95 gave a balance of both, faster convergence and lesser fluctuations.
References:

[1] Adaptive subgradient methods for online learning and stochastic optimization
J Duchi, E Hazan, Y Singer
Journal of Machine Learning Research 12 (Jul), 2121-2159

[2] ADADELTA: an adaptive learning rate method
MD Zeiler
arXiv preprint arXiv:1212.5701

e. To select the right values of **initialization**, **initial step length** η_0 and the **step function**, the following tuning was performed.

- For each of the possible three initialization, a zero vector, a vector of ones, a random vector containing values between 0 and 1, we created an elaborate table of performance of the optimizer given different timeouts and different values of η_0. This was done using a constant step function and a radically attenuating step function $\eta = \frac{\eta_0}{\sqrt{t}}$.

- Initializing α with a vector of ones resulted in an initial value far away from the global minima and hence a very slow descent of the original objective function was observed. Even the random initialization showed results similar to this. The best performance was obtained for a zero initialization which showed very fast convergence.

- Among the two possible step functions, the constant step function showed decent performance for $\eta_0 = 0.05 - 0.2$ but didn't perform well for smaller timeouts. So we chose $\eta = \frac{\eta_0}{\sqrt{t}}$ as our step function.

- Now, to select the optimal value of η_0, the following table was created to analyze the performance of the optimizer on each possible value.

η_0 \ Timeout(s)	5	10	15	20	25	30
0.5	5255.65	5234.27	5229.90	5228.62	5226.11	5225.57
1.0	5227.83	5227.65	5225.56	5226.25	5224.63	5225.67
1.5	5228.12	5226.65	5226.86	5225.06	5225.92	5225.29
2.0	5233.60	5231.91	5226.88	5224.52	5224.27	5224.96
2.5	5236.68	5231.05	5225.55	5228.95	5226.02	5225.01
3.0	5239.75	5234.40	5225.73	5225.63	5230.30	5227.86
3.5	5247.46	5249.41	5233.14	5228.24	5239.87	5233.63
4.0	5233.73	5228.93	5243.47	5232.07	5231.72	5229.24

From the table, it turns out that $\eta_0 = 1.0$ seems to perform the best for all given timeouts.

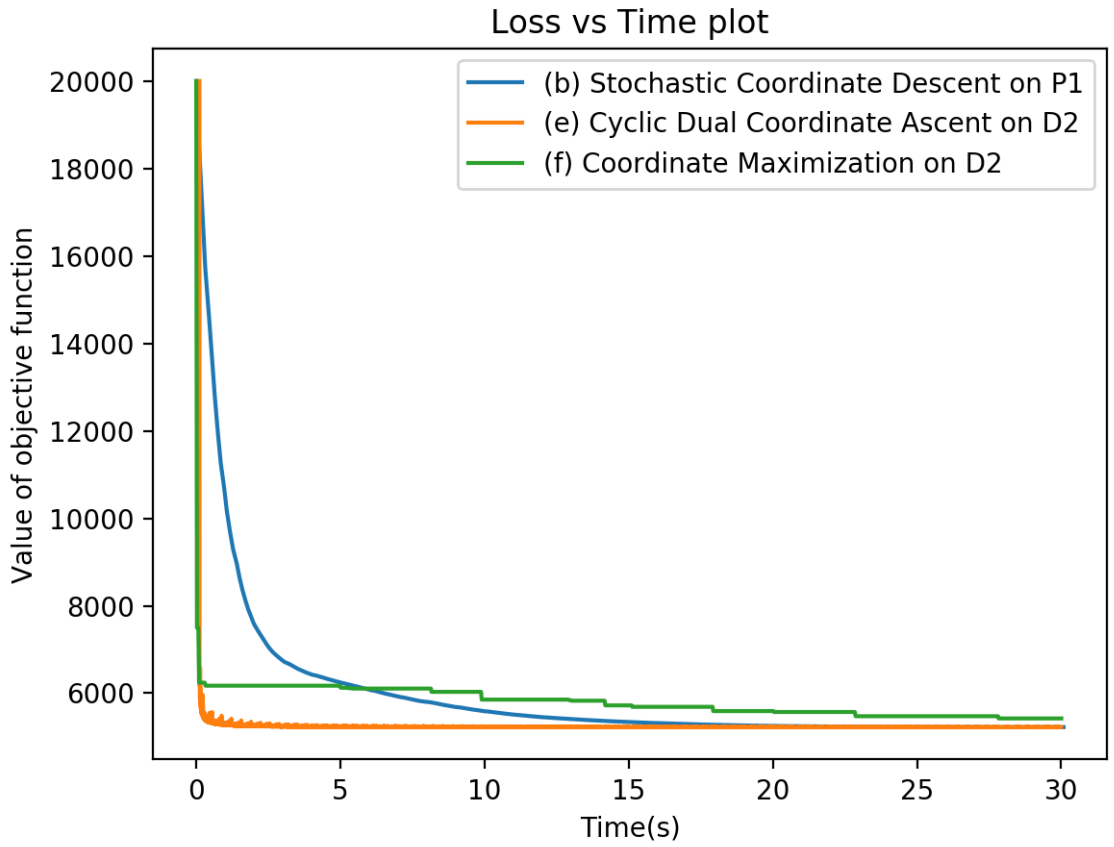
f. To select in which order to select coordinate for maximization, we created 2-fold validation sets. We tried out following orders:

- **Cyclic but deterministic (A):** We updated the coordinates in cyclic order from 1 to n.
- **Cyclic but random permutation (B):** We created a random permutation and moved in cyclic order along that permutation.
- **Uniform Random Coordinate (C):** We selected a coordinate randomly uniformly each time.

Since number of coordinates were very large, it was observed that **(C)** worked out to be the best method. The results are tabulated for complete dataset.

Method \ Timeout(s)	5	10	15	20	25	30
A	6172.21	6172.21	6172.21	6170.37	6170.37	6170.37
B	6482.75	6394.50	6215.85	6192.31	6145.23	6123.94
C	6504.06	6034.03	6008.64	5974.35	5971.57	5872.77

5.



Looking at the Objective function vs Time plot for all the three methods, we found that (b) Stochastic Coordinate Descent on P1 worked the best. We compared the methods across different values of C and different timeouts (for C = $[0.1, 0.6, .., 4.6]$ and timeout = $[10, 20, 30]$ seconds) and observed that (b) consistently performed better.
Out of 30 different combinations we tried, the odds in favour of (b) against (e) and (f) were $29 : 1 : 0$.

7. The problem is given by,

$$\boldsymbol{P'} \equiv \underset{\boldsymbol{w},\boldsymbol{\xi}}{\boldsymbol{argmin}} \left(\frac{1}{2}||\boldsymbol{w}||_2^2 + C \sum_{i=1}^{n} \xi_i^2 \right)$$

with the constraints

$$1 - \xi_i - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle \leq 0$$
$$-\xi_i \leq 0$$

The Lagrangian of $\boldsymbol{P'}$ will be given by

$$\boldsymbol{\mathcal{L}}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \left(\frac{1}{2} ||\boldsymbol{w}||_2^2 + C \sum_{i=1}^{n} \xi_i^2 + \sum_{i=1}^{n} \alpha_i (1 - \xi_i - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle) - \sum_{i=1}^{n} \beta_i \xi_i \right)$$

The dual of the above problem is given by

$$\boldsymbol{D'} = \underset{\boldsymbol{\alpha_i} \geq \boldsymbol{0}, \boldsymbol{\beta_i} \geq \boldsymbol{0}}{\boldsymbol{argmax}} \left(\underset{\boldsymbol{w}, \boldsymbol{\xi}}{\boldsymbol{argmin}} \left(\frac{1}{2} ||\boldsymbol{w}||_2^2 + C \sum_{i=1}^{n} \xi_i^2 + \sum_{i=1}^{n} \alpha_i (1 - \xi_i - y^i \left\langle \boldsymbol{w}, \boldsymbol{x^i} \right\rangle) - \sum_{i=1}^{n} \beta_i \xi_i \right) \right)$$

Differentiating the **Langrangian** with respect to $\boldsymbol{w}$, we get

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{w}} = 0 \implies \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y^i \boldsymbol{x}^i$$

Differentiating the **Langrangian** with respect to $\boldsymbol{\xi}$, we get

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\xi}} = 0 \implies \boldsymbol{\xi_i} = \frac{\alpha_i + \beta_i}{2C}$$

Eliminating $\boldsymbol{w}$ and ξ from the **Lagrangian**, we get

$$\equiv \underset{\boldsymbol{\alpha_i} \geq \boldsymbol{0}, \boldsymbol{\beta_i} \geq \boldsymbol{0}}{\boldsymbol{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle + C \sum_{i=1}^{n} \left(\frac{(\alpha_i + \beta_i)^2}{4C^2} - \frac{(\alpha_i + \beta_i)^2}{2C^2} \right) \right)$$
$$\equiv \underset{\boldsymbol{\alpha_i} \geq \boldsymbol{0}, \boldsymbol{\beta_i} \geq \boldsymbol{0}}{\boldsymbol{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \sum_{i=1}^{n} \frac{(\alpha_i + \beta_i)^2}{4C} \right)$$

Now, $\alpha_i + \beta_i \geq \alpha_i$ as $\alpha_i \geq 0$ and $\beta_i \geq 0$

$$\forall i, -\frac{1}{2C} (\alpha_i + \beta_i)^2 \leq -\frac{1}{2C} \alpha_i^2$$

$\therefore$ Maxima will occur when $\beta_i = 0, \forall i \in [n]$

$$\therefore \underset{\alpha_i \geq 0, \beta_i \geq 0}{\boldsymbol{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \sum_{i=1}^{n} \frac{1}{2C} (\alpha_i + \beta_i)^2 \right)$$

$$\equiv \underset{\alpha_i \geq 0}{\boldsymbol{argmax}} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^j \left\langle \boldsymbol{x^i}, \boldsymbol{x^j} \right\rangle - \sum_{i=1}^{n} \frac{1}{2C} \alpha_i^2 \right)$$

$$\equiv D2$$
$$\equiv P2$$

$\therefore$ Taking additional constraints has no effect.