Hw2

Kexin Feng 5165462

October 27, 2018

1 Problem 1:

1.1 Analysis and experiment report

The error rates at different C:

Table 1: Error rates

C	0.01	0.1	1	10	100
err	9.50E-03	1.45E-02	1.10E-02	1.10E-02	1.20E-02
error std	6.87E-03	9.60E-03	7.00E-03	6.63E-03	4.00E-03

(b) From Fig. 1, we can see the following:
(i)As C increases, the mean error rate doesn't show apparent change or tendency beyond the standard error.
(ii) As C increases $1/|w|$ only decreases at the beginning; after C>0.1, it basically stays the same.
(iii) The support(corresponding to $0 < a_n <= C$) number doesn't show significant change beyond the standard error, either. The number of marginal points(corresponding to $0 < a_n < C$) has also been shown in the fourth plot and it increases as C increases.
In order to evaluate these results and make sense out of them, we need to go back to see how this maximum margin classifier and slack variable works.
The training process is to maximize the following. We denote the whole data set $\mathcal{S}$, and its sub set $\mathcal{A}$

$$\max_w \left\{ \frac{1}{|w|^2} \min_{\mathcal{S}}(t_n y_n) \right\} \tag{1}$$

$$\xrightarrow{w \to \kappa w'} \max_w \left\{ \frac{1}{|w'|^2} \right\} \tag{2}$$

$$\text{where } \kappa = \min_{\mathcal{S}}(t_n y_n) \tag{3}$$

After introducing the slack variable, instead of maximizing the whole set's minimum distance(to the decision boundary), now we just need to maximize the minimum distance (to the decision boundary) of a subset of data. But also pay the price of having error term $\max(0, 1 - y_n t_n$. To put it equation,

$$\max_w \left\{ \frac{\min_{\mathcal{S}}(t_n y_n)/\kappa}{|w|^2/\kappa} \right\} \tag{4}$$

$$\xrightarrow{w \to \kappa w'} \max_w \left\{ \frac{\min_{\mathcal{S}}(t_n y_n)/\kappa}{|w'|^2} \right\} \tag{5}$$

$$\text{where } \kappa = \min_{\mathcal{A} \subset \mathcal{S}}(t_n y_n) \tag{6}$$

Lagrangian becomes

$$L = \frac{1}{2}|w|^2 + C \sum_n \xi_n \tag{7}$$

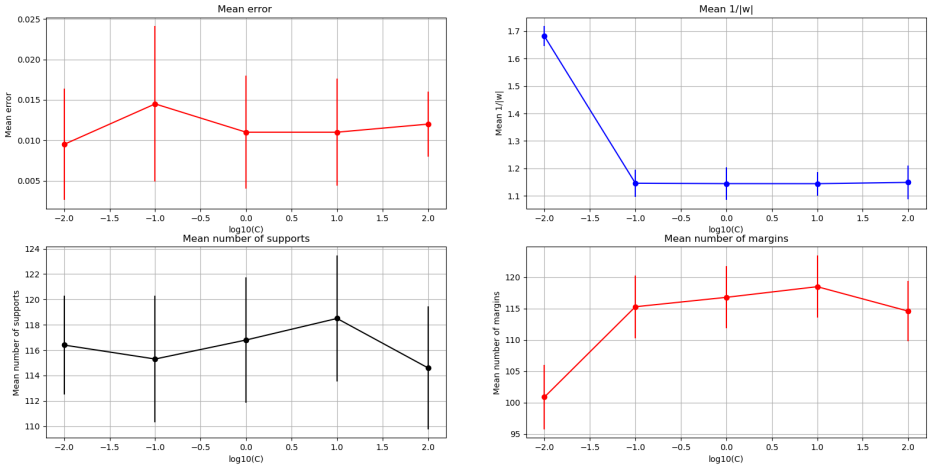

Figure 1: Performance of different C

So C is the regulation factor that penalizes error function $\sum_n \xi_n$. When C increases, the error tolerance $\sum_n \xi_n$ becomes small, and the distance of the margin points to the decision boundary, $d = \frac{1}{|w|^2}$ is smaller, and the capacity of the model is smaller.
So about result (i), when C increases, the capacity of the model is smaller, the error rate could increase because of under fitting, or could decrease due to over fitting. So it's possible to also have it basically no change.
About result (ii), the distance of the margin points to the decision boundary, $d = \frac{1}{|w|^2}$ decreases as predicted above.
About result (iii), since the tolerance for the error function decreases, we should see an decrease in the support number. But note that the direct effect of increasing C is the decrease of $\sum_n \xi_n$, which doesn't necessarily mean that the amount of non-zero ξ_n decreases. The value of $\sum_n \xi_n$ could decrease when the amount of non-zero ξ_n increases. So no wonder there is no change in the support number.
(c) After introducing the slack variable ξ_n, the Lagrangian becomes

$$L = \frac{1}{2}|w|^2 + C\sum_n \xi_n \tag{8}$$

Now the Lagrangian becomes a function of not just w,b, but also xi_n

$$L = L(w, b, \xi_n) \tag{9}$$

Also these new variables have constraint

$$\xi_n \geq 0 \tag{10}$$
$$t_n y_n(w, b) \geq 1 - \xi_n \tag{11}$$

where $y_n(w, b) = w^T \phi(x_n) + b$.
Note that the original definition of ξ is

$$\xi_n \geq |t_n - y_n| \, \mathbb{1}(t_n y_n < 1) \tag{12}$$
$$= (1 - t_n y_n) \, \mathbb{1}(t_n y_n < 1) \tag{13}$$
$$\mathbb{1}(I) = 1, \; x \in I \tag{14}$$
$$0, \; else \tag{15}$$

We can time is by t_n both sides, given $t_n = \pm 1$, to get

$$t_n y_n \geq 1 - \xi_n \tag{16}$$

Then on way is to use standard form for KKT calculation and treating ξ_n as a variable of the Lagrangian

$$L = C\sum \xi_n + \frac{1}{2}|w|^2 - \sum_n a_n(t_n y_n - 1 + \xi_n) - \sum_n \mu_n \xi_n \tag{17}$$

But there is another way to look at this Lagrangian. We can incorporate the constraint into the Lagrangian and treat ξ_n as a function of w and b, instead of a variable of the Lagrangian

$$L(w,b) = C\sum_n \xi_n(w,b) + \frac{1}{2}|w|^2 \tag{18}$$

$$= C\sum_n \max(0, 1 - t_n y_n) + \frac{1}{2}|w|^2 \tag{19}$$

$$\text{where } y_n = y_n(w,b) = w^T\phi(x_n) + b \tag{20}$$

So the constraints are satisfied.

$$\xi_n \geq 0 \tag{21}$$

$$t_n y_n(w,b) \geq 1 - \xi_n \tag{22}$$

But the margin constraint $t_n y_n(w,b) \geq 1$ is only satisfied when $\xi_n = 0$.

1.2 The algorithm

The algorithm is: first, do dimension reduction to remove all the columns that don't have any variance. Then based on the dual Lagrangian, run the KKT optimization package to get SVM parameters. Then based on the SVM parameters, discriminant function can be given, which is a kernel form $Y(x) = a_n t_n k(x_n, x) + b$. Finally run validation to get the error rate.

2 Problem 2:

2.1 Algorithm

This experiment implements two optimization algorithms and compare their performance.
The first one is Pepsos, which is a stochastic projection minibatch gradient descent. This algorithm, like other projection gradient descent algorithms, deals with non-smooth Lipschiz funciton, like $max0, 1 - x$. In this svm, the objective function is

$$\mathcal{L} = \sum_n max(0, 1 - y_n w^T x_n) + \frac{1}{2}\lambda|w|^2 \tag{23}$$

The details of this algorithm is in Figure 1. of paper [1].
The second one is to use a softplus function as a smooth version of the max function above, so as to use stochastic gradient descent algorithm. In order to compare with the first algorithm, here it's actually mini-batch SGD.

$$\nabla\mathcal{L} = \frac{\lambda}{2}|w|^2 - \frac{1}{n}\sum_n \frac{t_n x_n}{1 + \exp\left(-(1 - t_n w^T x_n)/a\right)} \tag{24}$$

Input: S, λ, T, k
Initialize: $w1$ = np.zeros
For t = 1,2,..., T
{
Set $\eta_t = \frac{1}{\lambda t}$
Randomly choose A_t a subset of S s.t. $|A_t| = k$
$w_{t+1} = w_t(1 - \eta_t\lambda) + \frac{\eta_t}{k}\sum_{A_t} \text{gradient}(w_t)$
}
return $w_T = \frac{1}{T}\sum w_t$

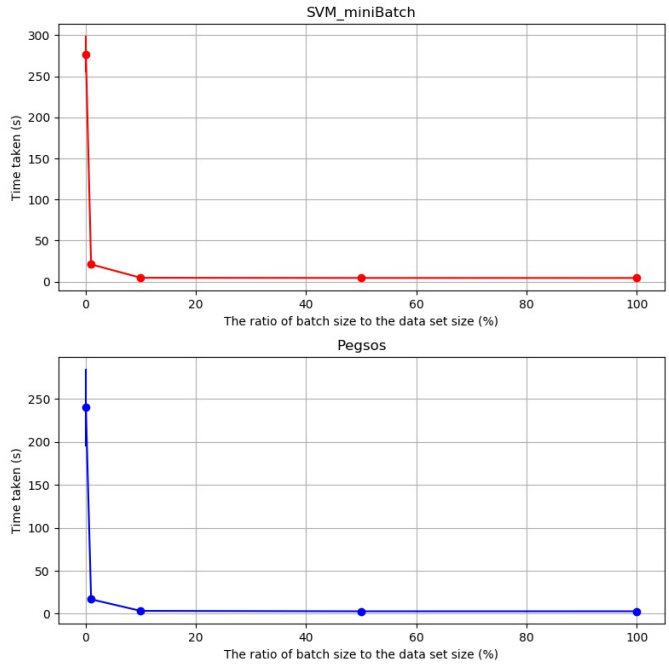

Figure 2: Time consumption for SGD and Pegsos

Here the gradient(w_t) is the second term in equation (24).

The termination condition for the two algorithms are both when the total # of gradient reaches ktot = 100*n

It's worthwhile to notice that when the batch size k = 1, both algorithm become usual stochastic gradient descent where only one data is pick out randomly at each step. When k = n, the data set size, they both become usual gradient descent where, at each iteration, the gradient is averaged over the whole data set. We can expect that the latter situation will be very inefficient, as we will see later.
The evaluation of the performance is based on convergence criteria, that is by comparing the decreasing of loss function w.r.t. time.

Table 2: Run times

batch size k	1	20	200	1000	2000
Pegsos	240.098	16.87	3.3838	2.8974	2.9132
std	44.67898578	0.188812076	0.147245373	0.160823817	0.117452969
SGD miniBatch	277	21.1	4.8	4.58	4.51
std	21.9	0.898	0.549	0.426	0.317

2.2 Experiment result and analysis

First of all, in table 2 it shows the run time required for different batch sizes. The corresponding plot is Fig. 2. We can see that the two algorithms behaves almost the same as far as the efficiency is concerned. Also as k decreases, the required time increases. This is because array batch mean calculation becomes for loop when batch size k becomes smaller.
Next, we have the primal objective function decreasing plots in Fig 2.1 and Fig. 2.1 I've taken log10 of it in order to see its order of magnitude change. The x axis is one unit per 2000 gradient calculations.
We can see that SGD converges faster than Pegsos when k = 1 and k = 20; SGD reaches asymptotic region at around 10$\hat{0}$.25, but Pegsos reaches asymptotic region at arount 10$\hat{0}$.5. But when k is 200, 1000 and 2000, Pegsos is better.

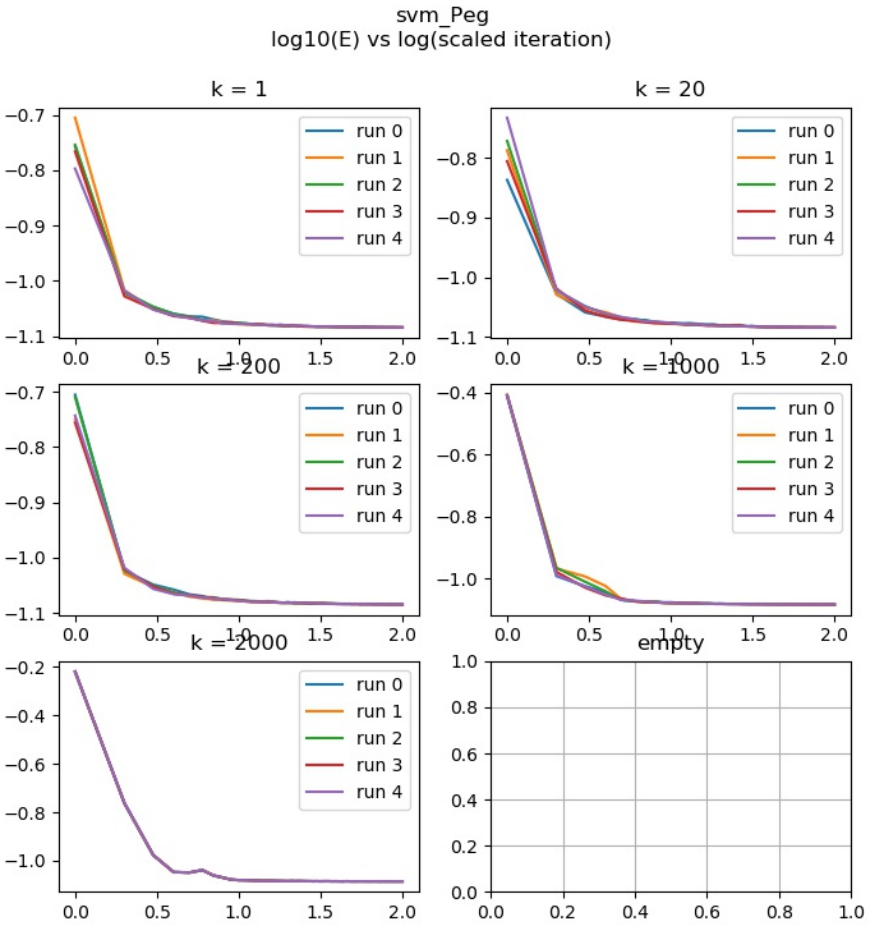

Figure 3: Pegsos loss function log-log plot

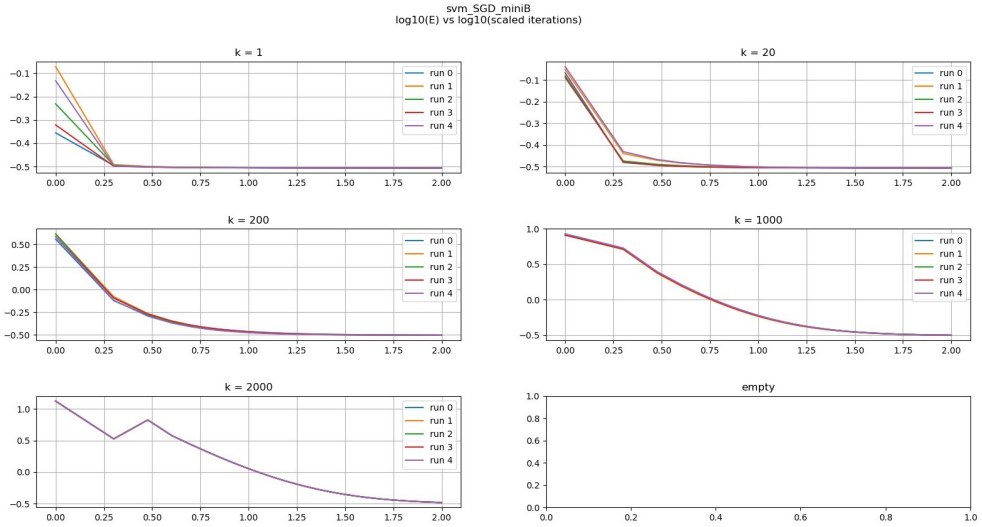

Figure 4: SGD mini-batch loss function log-log plot

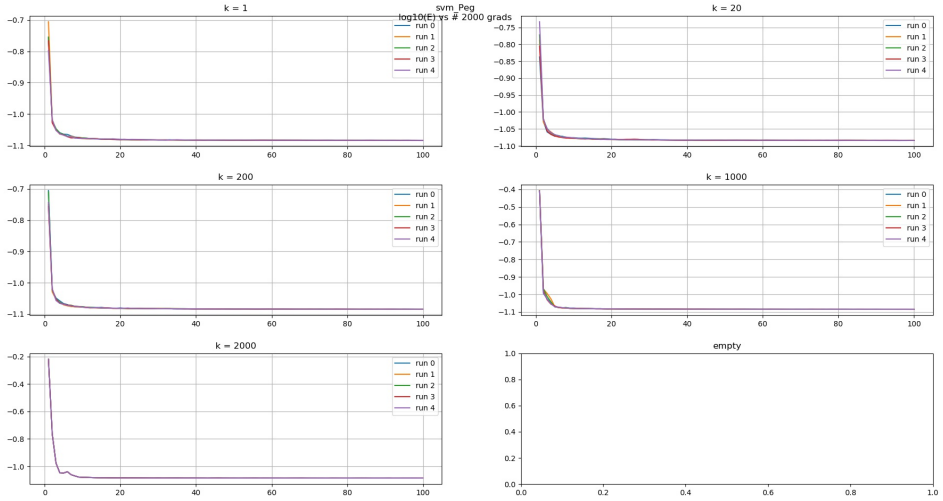

Figure 5: Pegsos loss function log-linear plot

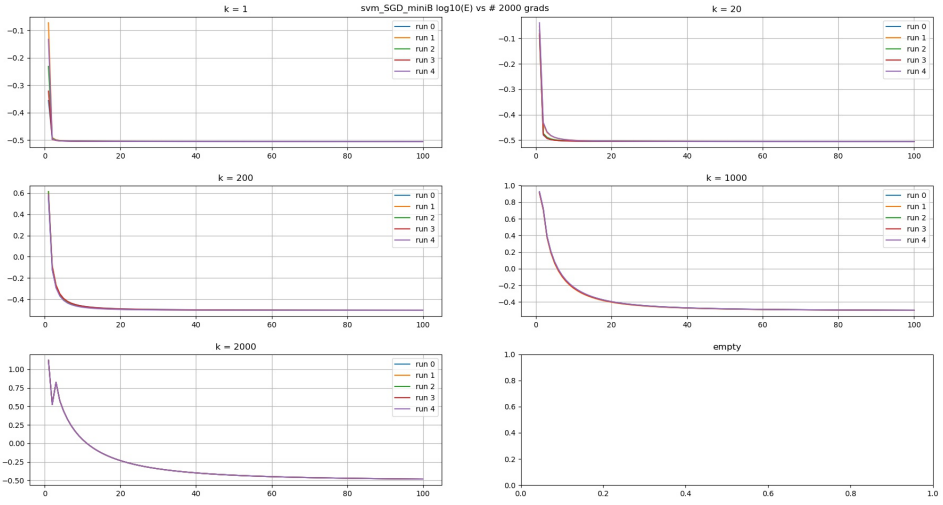

Figure 6: SGD mini-batch loss function log-linear plot

The reason could be that when batch size is small, it's close to the primal stochastic gradient descent. SGD utilize the smoothness of softplus function, but Pegsos deals with max. Because at each iteration, only a small fraction of data is taken to calculate the gradient, the smoothness of objective function is more important. That's why we see SGD behaves better. But when k, the batch size, grow bigger, calculating the batch average gradient is not very efficient regarding the converging speed. On the contrary, with a fixed total gradients, increasing batch size reduces the iterations. That's why we see when k increases SGD behaves worse. Pepsos has the similiar tendency, but since there is a projection operation to constrain it within a ball of radius R, it is not significantly affected by increasing of batch size. The merit of using mini batch is to reduce the variance so as to accelerate the convergence, but here we don't we see much such effect.

References

[1] S. ShalevShawtrz, Y. Singer, and N. Srebro. "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM,"