CSci5525 Homework 2

Rihan Chen

Oct 20, 2015

Problem 1

Problem 1(a)

We only need to prove that for $\{x_1 \dots x_n\}$.The K matrix, for each element K_{ij} is $K(x_i, x_j)$, is positive semidefinite.(The symmetric property can be proved by positive semidefinite as well) For $K_1 \dots K_m$ are all positive semidefinite(because their corresponding kernel function is valid), we can prove that their linear combination with nonnegative weights is also positive semidefinite. Suppose x is any factor with length n

Then, we have:

$$\begin{aligned} x^T Kx &= x^T(\textstyle\sum_{j=1}^{m} w_j K_j)x \\ &= w_1 x^T K_1 x + w_2 x^T K_2 x + \ldots + w_m x^T K_m x \\ &\geq 0 \end{aligned}$$

This is because that as each K_i is positive semidefinite, $x^T K_i x$ is nonnegative. And w_i is also nonnegative.Therefore, we prove that K is positive semidefinite.

Problem 1(b)

We only need to prove that for $\{x_1 \dots x_n\}$.The K matrix, for each element K_{ij} is $K(x_i,x_j)$, is positive semidefinite.(The symmetric property can be proved by positive semidefinite as well) As K_1 and K_2 is positive semidefinite, therefore we can use eigendecomposition to express them as:

$$K_1 = \sum \mu_i m_i m_i^T$$

$$K_2 = \sum \nu_j n_j n_j^T$$

where the scalars μ and ν are the eigenvalues, the vector is their corresponding eigenvector. As the matrices are both positive semidefinite, the eigenvalues are all nonnegative.

Then the Hadamard product can be expressed as follow:

$$\begin{aligned} K &= K_1 \odot K_2 \\ &= \sum_{ij} \mu_i \nu_j (m_i m_i^T) \odot (n_j n_j^T) \\ &= \sum_{ij} \mu_i \nu_j (m_i \odot n_j)(m_i \odot n_j)^T \end{aligned}$$

Then, we prove that for each $(m_i \odot n_j)(m_i \odot n_j)^T$ is positive semidefinite. Suppose x is any vector with length n. Then we have:

$$x^T (m_i \odot n_j)(m_i \odot n_j)^T x = (\sum_k m_{i,k} n_{j,k} x_k)^2$$

which is nonnegative apparently. Hence we prove that the K matrix is positive semidefinite.

Problem 1(c)

We only need to prove that for $\{x_1 \dots x_n\}$..The K matrix, for each element K_{ij} is K(x_i,x_j), is positive semidefinite.(The symmetric property can be proved by positive semidefinite as well). First of all we need to show that the linear kernel is the valid the kernel function, the kernel is $K_{linear}(x_i, x_j) = x'_i x_j$ The corresponding K_linear matrix is obviously positive semidefinite,since it actually can be expressed as AA^T, which is positive semidefinite. Suppose x is any compatible vector, then we have:

$$x^T AA^T x = (A^T x)^T (A^T x)$$

which is actually the sum of square number, which is nonnegative. Therefore, K_linear matrix is obviously positive semidefinite.

Also, $K_{linear^*}(x_i, x_j) = x'_i x_j + 1$ is valid, since the corresponding K_{linear^*} matrix is the sum of K_linear and matrix with all elements equal to one, both of them are positive semidefinite matrix, hence their combination with nonnegative weights is also positive semidefinite,which has been proved in (a).

The kernel in this question is $x'_i x_j + 1$ to the power of 2015, which can be taken as the Hadamard product of 2015 valid kernel functions. Hence the kernel function is valid by (b)

Problem 1(d)

Firstly, the kernel can be written as follow:

$$\begin{aligned} K(x, x') &= \exp(-\tfrac{1}{2}\|x - x'\|^2) \\ &= \exp(-\tfrac{1}{2}\|x\|^2)\exp(\tfrac{1}{2}x^T x')\exp(-\tfrac{1}{2}\|x'\|^2) \end{aligned}$$

Secondly, as $\exp(-\frac{1}{2}\|x\|^2)$ is actually a function of x. therefore we first prove that

$$K^*(x, y) = f(x)K^{**}(x, y)f(y)$$

$K^*(x, y)$ is a valid kernel if $K^{**}(x, y)$ is valid and $f(x)$ is a mapping from R^n to R This is simple, by definition. Any kernel is:

$$K^{**}(x, y) =< \Phi(x), \Phi(y) >$$

, where $\Phi(x)$ is a mappping from R^n to R Therefore, we can define a mapping as $f(x)\Phi(x)$. Then we prove the $K^*(x, y)$ is a kernel. Secondly,we prove that $K^{scale}(x, y) = cK(x, y)$ is a valid kernel if c is some positive number and K(x,y) is valid. It is easy to show, becacuse K matrix corresponding to the valid kernel function is positive semidefinite. hence cK is positive definite as well.

Next, we prove that $\exp(K(x, y))$ is a vaild kernel function if $K(x, y)$ is valid. By taylor expansion we have

$$\exp(K(x, y)) = \lim_{n\to\infty} \sum_0^n \frac{1}{n!} K(x, y)^n$$

As we already prove that the *sum*, *Hadamard* and *scaling* of valid kernel is also valid. Hence, we get the conclusion.

Finally, we apply this conclusion to gaussian kernel function,

$$\exp(K(x, y)) = \exp(\frac{1}{2}(x^T x'))$$

$$f(x) = \exp(-\frac{1}{2}\|x\|^2)$$

The linear kernel is valid proved in (c), then we can apply the conclusion above. Finally, we prove that the gaussian kernel is valid.

Problem 2

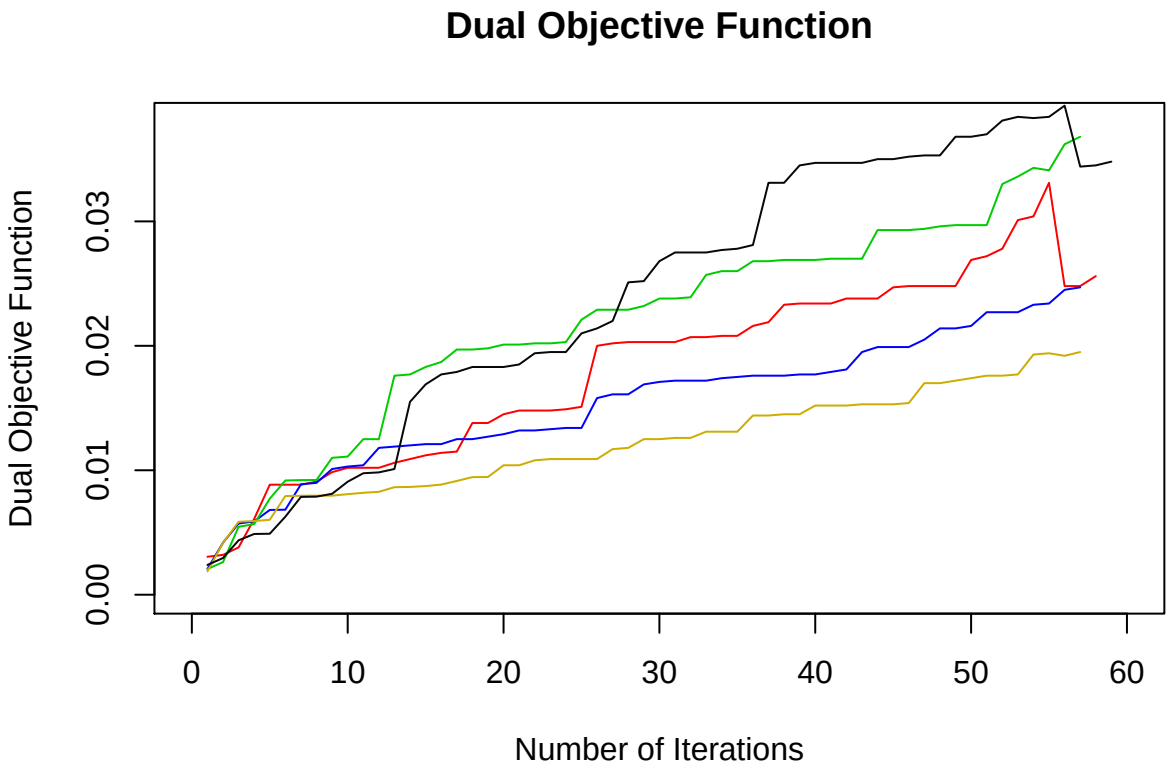
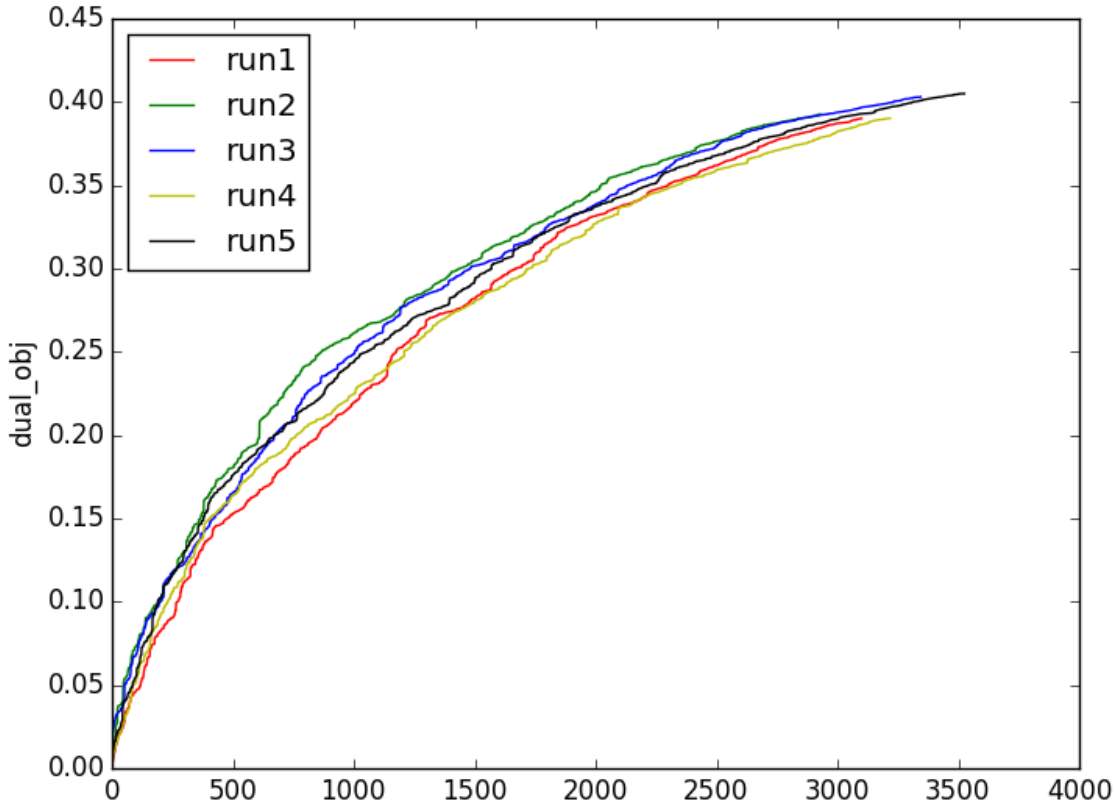


Table 1: Summary of Runtime for SMO

Avg_Runtime	3.70266
Std_Runtime	0.72561

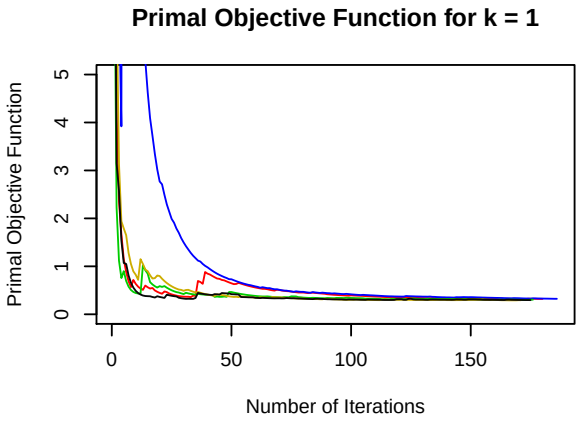
From the plot, we see that the the dual objective function is increasing when the number of iteration increasing in general. Sometimes, the objective function is dropping down, it may be due to the randomness of the algorithm. The number of iterations in our algorithm is actually efficient iteration, it refers to the iteration that the update of lagrange multiplier actually happens. Becuase we add some constraints in our algorithm in order to speed up the codes, iterations, hence, are not always useful. In other words, we may jump out of a iteration directly, if any of the constraints is violated. We build the model by simplified SMO model. For the first α_i, we go through the entire observations (in this case is 2000), then we choose α_j which j is not equal to i. The sub-problem can be solved efficiently because SMO doesn't need to any extra matrix storage and no iterative numerical routine for each sub-problem. Actually, for each iteration, the SMO just simply optimize the smallest optimization problem. Because the solution should satisfy KKT condition, the smallest optimization is actually two Lagrange multipliers. For the smallest subproblem, no matrix computation is involved, the two Lagrange multiplier can both get their closed form solution in each iteration. As for each time, we are faced with simplest QP problem, we can speed up the algorithm. Technically, in this case, we will always choose the Lagrange multipliers that between 0 and C. With this thershold, we are no longer using the brute force method to select the second multiplier. Moreover, before we update the multipliers, we check several constraints. If any of the constraint is violated, no further update is required, therefore, we proceed into next loop directly without any meaningless updating. To be more specific, in our algorithm, we almost prevent us from any unnecessary loop. For example, for calculating the objective function each time, we implement matrix multiplication instead of double loop in order to speed up our code. However, the

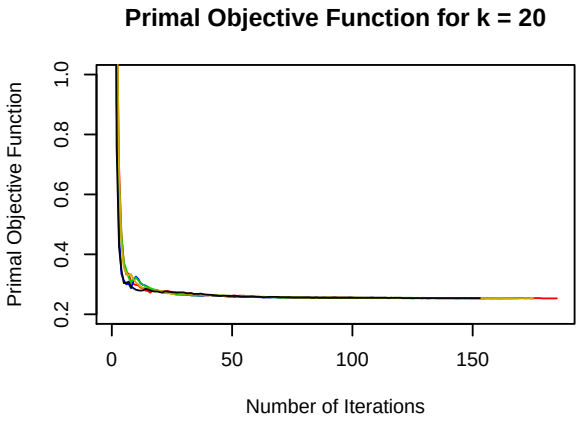
drawback for the simplied SMO model is that the choice of a_i and a_j is not heuristic. Therefore, it cannot fully optimize the dual objective function within 10000 iteration. We display the the fully optimize model as follow:

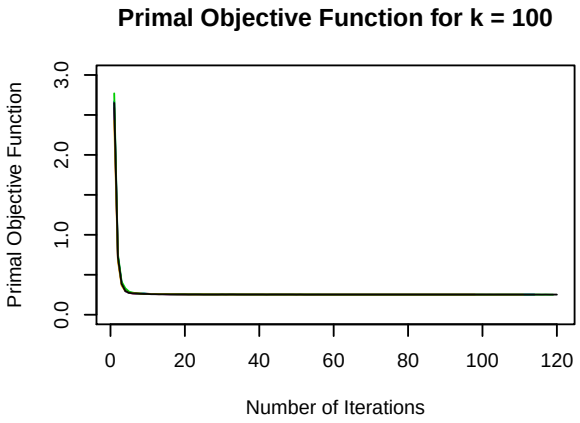


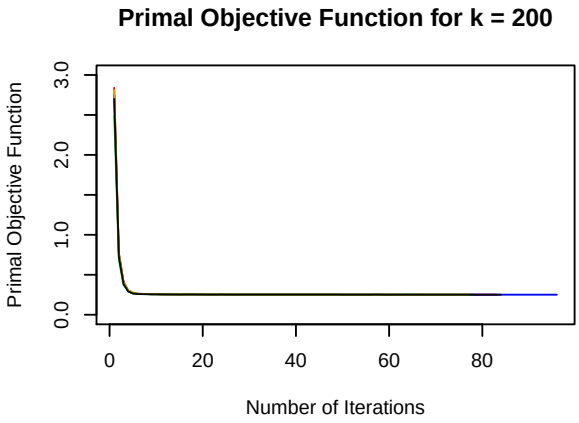
From the plot we see that for getting the maximum for the dual objective function, we need to have 3500 or above effective iterations. It requires quite a long time.

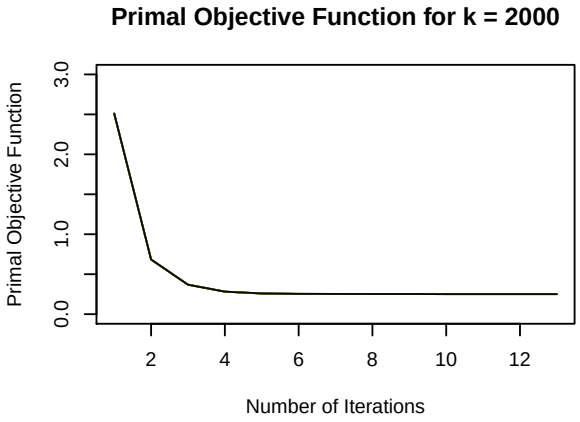
Problem 3


Primal Objective Function for k = 1
Primal Objective Function
0
1
2
3
4
5
0
50
100
150
Number of Iterations


Primal Objective Function for k = 20
Primal Objective Function
0.2
0.4
0.6
0.8
1.0
0
50
100
150
Number of Iterations


Primal Objective Function for k = 100
Primal Objective Function
0.0
1.0
2.0
3.0
0
20
40
60
80
100
120
Number of Iterations


Primal Objective Function for k = 200
Primal Objective Function
0.0
1.0
2.0
3.0
0
20
40
60
80
Number of Iterations


Primal Objective Function for k = 2000
Primal Objective Function
0.0
1.0
2.0
3.0
2
4
6
8
10
12
Number of Iterations

From the plot, we see that the stochastic gradient method has greater variability when the k is small in terms of the value for primal objective function. When k gets larger, the method has less variability. Moreover, when k =2000, their is no stochastic at all. What's more, larger minibatch sizes converge rapidly than small ones. This is not to say that larger k gets rapid speed but that larger k is more easily to achieve the converge requirement in our problem. The summary for the runtime is as follow:

Table 2: Summary of Runtime for SGD

size	Avg	Std
k = 1	0.72561	0.00723
k = 20	0.63359	0.03331
k = 100	0.56003	0.02691
k = 200	0.53009	0.02066
k = 300	0.37379	0.04723