FASTER COORDINATE DESCENT VIA ADAPTIVE IMPORTANCE SAMPLING

Liu Zhihan[1,*] **Xu Xin** [1,*]

January 3, 2021

1 Task Distribution

- Liu Zhihan: PB18051211
  Responsible for implementing the sampling algorithms and slides producing.
- Xu Xin: PB18000042
  Responsible for constructing the coordinate descent part, result plotting, and trying svm.

2 Introduction

2.1 Problem Background

Coordinate descent is a frequent used tool in the optimization for its higher space and computational efficiency especially compared with gradient descent.

Like we can use stochastic gradient descendent(SGD) to replace gradient descendent(GD) for the reason that GD needs to train on the whole data for one iteration while SGD only use one or a subset of data. And it practice it also proves that using SGD is much faster.

For the same reason, we can substitute the coordinate-wise coordinate descent with stochastic coordinate descent, which selects the updating dimension uniformly at random.

However, there exists the situation in which the traditional stochastic coordinate descent with uniform sampling (we call it Uniform in the text later) is not efficient. For example, if true α =(1,1,0), while the present iterative α = (0,1,0), if we always sample the second or the third coordinate with a probability of $2/3$, then we need more than N iterations with a probability of $(2/3)^N$. But if we use coordinate-wise coordinate descent we may arrive at our optimal point just for one iteration. So a new sampling strategy is in need to improve the convergence speed of stochastic coordinate descent.

2.2 Preliminaries

To help establish our method, we introduce some useful definitions here.

Definition 1 *Our Primal-dual optimizer target pair:*

$$\begin{array}{l}\min_{\boldsymbol{\alpha}\in\mathbb{R}^n}\left[\mathcal{O}_A(\boldsymbol{\alpha}) := f(A\boldsymbol{\alpha}) + \sum_i g_i\left(\alpha_i\right)\right] \\ \min_{\boldsymbol{w}\in\mathbb{R}^d}\left[\mathcal{O}_B(\boldsymbol{w}) := f^*(\boldsymbol{w}) + \sum_i g_i^*\left(-\boldsymbol{a}_i^\top \boldsymbol{w}\right)\right]\end{array}$$

where we have $A = \left[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\right], w := (A\boldsymbol{\alpha})$

Definition 2 *Duality Gap and Coordinate-wise duality gaps:*

$$G(\boldsymbol{\alpha}, \boldsymbol{w}) := \mathcal{O}_A(\boldsymbol{\alpha}) - \left(-\mathcal{O}_B(\boldsymbol{w})\right)$$

$$G(\boldsymbol{\alpha}) = \sum_i G_i(\alpha_i) := \sum_i \left(g_i^* \left(-\boldsymbol{a}_i^\top \boldsymbol{w} \right) + g_i(\alpha_i) + \alpha_i \boldsymbol{a}_i^\top \boldsymbol{w} \right)$$

Definition 3 *Dual residual:*

$$\kappa_i^{(t)} := \min_{u \in \partial g_i^* \left(-\boldsymbol{a}_i^\top \boldsymbol{w}^{(t)} \right)} \left| u - \alpha_i^{(t)} \right|$$

And we can define an important term for our further analysis which effect the convergence speed.

Definition 4

$$F^{(t)} := \frac{1}{n^2 \beta} \sum_{i \in I_t} \left(\frac{\left| \kappa_i^{(t)} \right|^2 \|\boldsymbol{a}_i\|^2}{p_i^{(t)}} \right)$$

where $F^o$is a upper bound of $\mathbb{E}F^{(t)}$

3 Methodology

3.1 Motivation

Non-Uniform sample strategy Rather take a Uniform sampling strategy, we should be adopt a Non-Uniform sample strategy which need satisfy:

- Higher probability on those dimensions where the gap or the loss is relatively bigger and we can value the gap
- Good theoretical Guarantee on the convergence speed
- Be able to have a generalized form facing different optimization problem

To value the gap, we use two definitions described in the previous section 2.2, dual gap and dual residual and derive two algorithms respecitvely using one of them as the value of the gap of each coordinate.

3.2 Gap-wise sampling

We desire to sample each coordinate according to its duality gap.

$$p_i^{(t)} := \frac{G_i\left(\boldsymbol{\alpha}^{(t)}\right)}{G\left(\boldsymbol{\alpha}^{(t)}\right)}$$

3.3 Adaptive sampling

We turn to consider another bound to minimize:

$$T \geq \frac{5F^\circ n^2}{\varepsilon} + \frac{5\varepsilon_A^{(0)}}{\varepsilon p_{\min}}$$

We want to minimize F^o and $1/p_{min}$ at the same time. So we have following method to choose:

- adaptive

$$p_i^{(t)} := \frac{\left| \kappa_i^{(t)} \right| \|\boldsymbol{a}_i\|}{\sum_j \left| \kappa_j^{(t)} \right| \|\boldsymbol{a}_j\|}$$

- Support Set uniform

$$p_i^{(t)} := \begin{cases} \frac{1}{m_t}, & \text{if } \kappa_i^{(t)} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

here m_t is the cardinality of the support set of $\kappa^{(t)}$

- ada-uniform sampling

$$p_i^{(t)} := \begin{cases} \frac{\sigma}{m_t} + (1-\sigma)\frac{\left|\kappa_i^{(t)}\right| \|\boldsymbol{a}_i\|}{\sum_j \left|\kappa_j^{(t)}\right| \|\boldsymbol{a}_j\|}, & \text{if } \kappa_i^{(t)} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

3.4 Adaptive Sampling Coordinate Descent

We have a framework for adaptive sampling coordinate descent, the pseudo code is as follows:

Algorithm 1 Coordinate Descent

Let $\boldsymbol{\alpha}^{(0)} := \mathbf{0} \in \mathbb{R}^n, \boldsymbol{w}^{(0)} := \boldsymbol{w}(\boldsymbol{\alpha}^{(0)})$
for $t = 0, 1, ..., T$ **do**
  Sample $i \in [n]$ randomly according to $\boldsymbol{p}^{(t)}$
  Find $\Delta\alpha_i$ minimizing $\mathcal{O}_A(\boldsymbol{\alpha}^{(t)} + \boldsymbol{e}_i \Delta\alpha_i)$
  $\boldsymbol{\alpha}^{(t+1)} := \boldsymbol{\alpha}^{(t)} + \boldsymbol{e}_i \Delta\alpha_i$
  $\boldsymbol{w}^{(t+1)} := \boldsymbol{w}(\boldsymbol{\alpha}^{(t+1)})$
end for

4 Experiment

4.1 Data processing

We call the ReadLines function to get the data used in this article: Mushrooms and ionosphere data is read from the corresponding website directly, and we import rcv1 data by downloading binary data .

Due to the large rcV1 data set, we adopted the method in the paper: randomly select a certain number of rows and columns, and finally remove zero rows and zero columns.However, we extracted fewer rows and columns than in the article, because it would be time-consuming to perform gradient descent later when the dimension was higher.

4.2 Lasso Problem

4.2.1 Empirical Setting for Lasso

For Lasso Problem, our specific formulas are:

$$f(A\boldsymbol{\alpha}) := ||A\boldsymbol{\alpha} - y||^2 \quad g(\boldsymbol{\alpha}) := ||\boldsymbol{\alpha}||_1$$

For we assume g is Lipschitz continuous before, we use Lipschitz trick here:

$$\bar{g}_i(\alpha_i) := \begin{cases} \lambda |\alpha_i|, & \text{if } |\alpha_i| \leq B \\ +\infty, & \text{otherwise} \end{cases}$$

Here we set $B = \frac{1}{\lambda}\left(f\left(A\boldsymbol{\alpha}^{(0)}\right) + \lambda \left\|\boldsymbol{\alpha}^{(0)}\right\|_1\right)$

And the conjugate of $\hat{g}$:

$$\bar{g}_i^*(u_i) = \max_{\alpha_i : |\alpha_i| \leq B} u_i\alpha_i - \lambda|\alpha_i| = B\left[|u_i| - \lambda\right]_+$$

We test our algorithm which is based on sub-gradient coordinate descent on two data sets *mushrooms* and *rcv1*.

4.2.2 Empirical Setting for Simulations

- Together with the SVM problem, We do the simulations on three data sets and the magnitude of the data sets are displayed here.

Dataset we use	d	n
mushrooms	112	8124
rcv1*	300	8000
ionosphere	351	33

- On the prepossessing step, we deleted the blank columns and rows.
- For Lasso problems with first two data sets, we choose $\lambda = 0.05$

4.2.3 Code for Lasso

We only list some core code here, the whole code is available in the .rmd. The parameters in the following functions are almost the same name as we mentioned in 4.4.1, so we do not explain them anymore.

```
lasso.w_func <- function(alpha,A,y){
  return(2*(A %*% alpha - y))
}
```

```
lasso.gap <- function(alpha,lambda,B,A,dimension,y){
  # 计算对偶gap
  i <- dimension
  w <- lasso.w_func(alpha,A,y)
  gap <- B*positive(abs(sum(A[,i]* w))-
lambda)+lambda*abs(alpha[i])+alpha[i]*(sum(A[,i]* w))
  return(gap)
}
```

```
lasso.subgrad <- function(alpha,lambda,A,dimension,y,...){
  #返回次梯度
  i <- dimension
  n <- length(alpha)
  # First calculate the subgrad of |alpha|, sub1
  if (alpha[i] > 0 ){
    sub1 <- lambda
  }
  else if(alpha[i] < 0){
    sub1 <- -lambda
  }
  else {
    seed <- 0
    sub1 <- lambda*seed
  }
  # Then calculate the grad of ||A\alpha -y||^2
  term <- 2*(A%*%alpha-y)
  term <- as.vector(term)
  sub2 <- sum(A[,i]*term)
  subgrad = sub1 + sub2
  return(subgrad)
}
```

```
lasso.dualres <- function(alpha,lambda,A,dimension,y, B){
  i <- dimension
  flag <-  0
  eps <- 1e-5
  w <- lasso.w_func(alpha, A, y)
  input <- -t(A[,i])%*%w
  if (input <= -lambda - eps){
    g_sub <- -B
  }
  else if(input >= lambda + eps){
    g_sub <- B
  }
  else if(input > -lambda + eps & input < lambda + eps){
    g_sub <- 0
  }
  else if(input < 0){#g_sub is an interval
    g_sub_right <- 0
    g_sub_left  <- -B
    flag <- 1
  }
  else{
    g_sub_right <- B
    g_sub_left  <- 0
    flag <- 1
  }
```

```
  if (flag == 0){
      return (abs(alpha[i]-g_sub))
  }
  if (alpha[i] >= g_sub_left & alpha[i] <= g_sub_right){
    return (0)
  }
  else {
    return (min(abs(alpha[i]-g_sub_right),abs(alpha[i]-g_sub_left)
))
  }
}
```

```
p.ada.gap <- function(alpha,lambda,B,A,y){
  n <- length(alpha)
  p <- numeric(length = n)
  for (i in 1:n){
    p[i] <- lasso.gap(alpha, lambda, B, A, i, y)
  }
  psum <- sum(p)
  for (i in 1:n){
    p[i] <- p[i]/psum
  }
  return(p)
}
```

```
p.imp <- function(alpha,A,y,lambda,B){
  n <- length(alpha)
  p <- numeric(length = n)
  for (i in 1:n){
    p[i] <- norm(as.matrix(A[,i]),type = "F")
  }
  p = p/sum(p)
  return(p)
}
```

```
p.ada.uniform <- function(alpha, lambda, A, y, sigma,B){
  n <- length(alpha)
  p <- rep(1,n)
  second_term <- numeric(length = n)
  eps <- 1e-5
  m <- n
  for (i in 1:n){
    k <- abs(lasso.dualres(alpha, lambda, A, i,y, B))
    if( k < eps){
      p[i] <- 0
      m <- m-1
    }else{
      second_term[i] <- k*norm(as.matrix(A[,i]),type ="F")
    }
  }
  second_term <- second_term/sum(second_term)
  for(i in 1:n){
    if(p[i]==1){
      p[i] = sigma/m
    }
  }
  p <- p + second_term*(1-sigma)
  return(p)
}
```

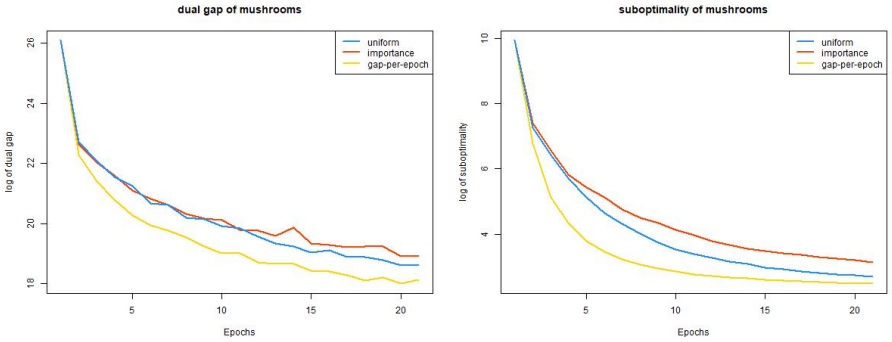
```
CD_each_iter <- function(sample_p,A,y,lambda,step0 = 0.1,...)
  {
  record_length <- 25
  epoch <- dim(A)[2]
  n <- dim(A)[2]
  alpha <- numeric(n)
  max_iter <- epoch*record_length
  record.gap <- numeric(length=record_length)
  record.loss <- numeric(length=record_length)
  gap <- numeric(n)
  B <- (norm(A%*%alpha - y,type = "2"))^2/lambda + sum(abs(alpha))
  w <- lasso.w_func(alpha,A,y)
  iter <- 0
  while(iter < max_iter){
  iter <- iter+1
  i <- sample(1:n,size = 1,prob = sample_p(alpha =
alpha,A=A,y=y,lambda=lambda,B=B,...))
  update <- lasso.subgrad(alpha = alpha,dimension =i ,A =A,y=y,lambda = lambda)
  inverses <- max(2*sum(A[,i]**2),1)
  alpha[i] <- alpha[i] - step0*update/inverses
  if(iter%%epoch==0){
    k <- iter/epoch
    for(j in 1:n){
      gap[j] <- lasso.gap(alpha, lambda, B, A, j,y=y)
    }
    record.gap[k] <- sum(gap)
    record.loss[k] <- lasso.loss(A, alpha, y)
  }
  }
  result <- list(dual.gap = log(record.gap), suboptimality = log(record.loss))
  return (result)
}
```

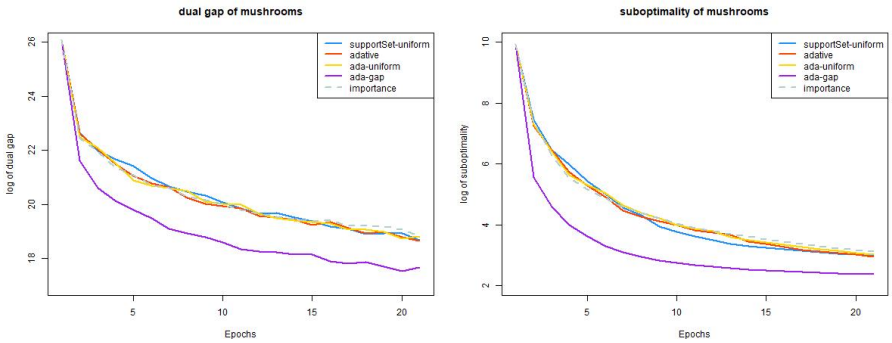
4.2.4 Result for Lasso in *mushrooms*

To see the performance of each method, we need to run several times to get the mean, which is very time-consuming when the data set is large.So we use a function to save the result in csv for later use.

We record suboptimality and duality gap as the main measures of algorithm performance. All reported results are averaged over 5 runs of each algorithm.

```
save_result <- function(CD, name, ...){
  result.gap <- matrix(0,nrow = 5, ncol = 25)
  result.loss <- matrix(0, nrow = 5, ncol = 25)
  for (i in 1:5){
    result <- CD(...)
   result.gap[i,] <- result$dual.gap
   result.loss[i,] <- result$suboptimality
  }
  name1 <- paste0("result/", name, "_gap.csv")
  name2 <- paste0("result/", name, "_loss.csv")
  write.csv(as.data.frame(result.gap),file = name1)
  write.csv(as.data.frame(result.loss),file = name2)
}
```





In the first figure, we can see that the gap-per-epoch method shows obvious advantages and its curve starts to decline rapidly, which is consistent with the results in the paper.However, in our results, the distinction between the two methods of importance and uniform is not obvious, and even the situation of uniform is better than that of importance, which is inconsistent with the paper.

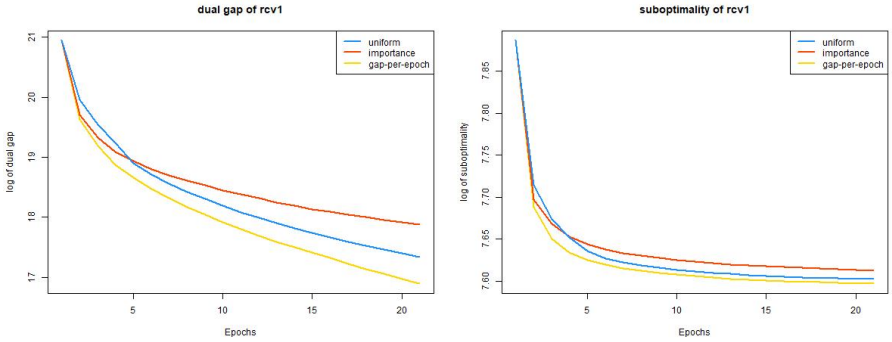
This is because we adopted Newton tangent method in the algorithm of coordinate descent, that is, the inverse of the second derivative is taken as the step length, which is a change for the traditional uniform method, and considering that the coordinate descent method is very sensitive to step size selection, it is not surprising that we get this result.

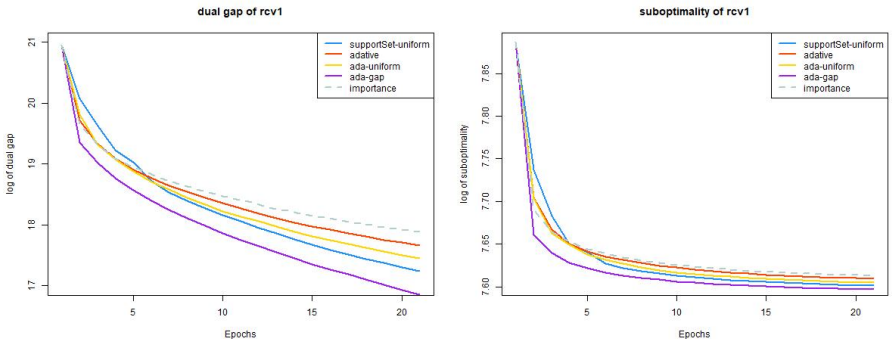
In the second figure, we can find that Ada-Gap method has obvious advantages, and all the curves have a certain degree of oscillation, which is consistent with the results in the original text. However, in our results, other new methods other than Ada-gap are just as good as the traditional imptance method, which is not consistent with the original result.

However, in the course of debugging, we were surprised to find that both the dual residuals and the Dual Gap are large, by several orders of magnitude larger than the ones in this article (we'll talk more about this later). Since ada-Gap method looks at the relative size of each component of Dual Gap, while ada-uniform, Supportset-uniform, and adative methods depend on the absolute size of the residual, the latter three methods are of no use in the case that the residual is much larger than zero (that is, the real situation when we reproduce it).

This explains why the Ada-Gap method has a distinct advantage while the other three new methods are similar to the traditional one. What's more, in the case of the data set, the result curve has a certain degree of concussion, so it's almost indistinguishable except for ada-Gap.

4.2.5 Result for for Lasso in *rcv1*

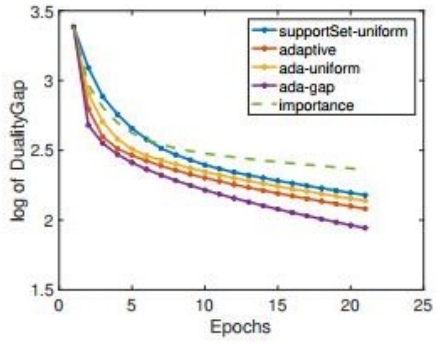




Compared with the data set musrooms, the curve obtained on the data set rcv1 is smoother, which is consistent with the original text. The reason for this result is similar to the analysis in the mushrooms data set, so I will not repeat it here. But it is worth mentioning that in the second picture, because the curve is very smooth, we can clearly see the pros and cons of each method before, and the result is very similar to the original text, except for the ada-gap method. The performance ranking of the three new methods is different from the text.

It is because when we calculate the dual residual, when the dual residual is less than a given relatively small number epsilon, it is considered to be zero (we take 1e-5), which will make the dual The calculation of the residual support is slightly different from the text (performance will change with the selected epsilon).

4.2.6 Problems in Lasso



However, under the initial conditions given in the article, the initial dual gap that we got is much larger. After we set some breakpoints to see value of some variables, we assume that the initial dual gap can not be so small as the article's. We calculate initial dual gap of mushrooms as follows:

```{r intial gap and loss of mushrooms}
B <- (norm(y.mushrooms,type = "2"))^2/0.05
gap0 <- 0
for (i in 1:112){
  gap0 <- gap0 + lasso.gap(numeric(112),0.05,B,A.mushrooms,i,y.mushro
oms)
}
gap0 <- log(gap0)
cat("B is",B, "\nlog of dual gap is",gap0)
```

```
B is 414960
log of dual gap is 26.09355
```

```r
```{r intial gap and loss of rcv1}
B <- (norm(y.rcv1,type = "2"))^2/7e-4
gap0 <- 0
for (i in 1:dim(A.rcv1)[2]){
  gap0 <- gap0 + lasso.gap(numeric(dim(A.rcv1)[2]),7e-4,B,A.rcv1,i,y.
rcv1)
}
gap0 <- log(gap0)
cat("B is",B, "\nlog of dual gap is",gap0)
```
```

```
B is 3801429
log of dual gap is 20.95059
```

4.3 Hinge-Loss SVM Problem

4.3.1 Empirical Setting for SVM

For SVM Problem, our specific formula is

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \mathcal{O}_A(\boldsymbol{\alpha}) := \frac{1}{n}\sum_{i=1}^{n}\varphi_i^*\left(-\alpha_i\right) + \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i \boldsymbol{a}_i\right\|_2^2$$

where $\varphi_i(s) := [1 - sy_i]_+$

We test our algorithm which is based on sub-gradient coordinate descent on one data set *ionosphere*. The magnitude of the dataset is desribed before and we set $\lambda = 7 \times 10^{-4}$ here.

4.3.2 Code for SVM

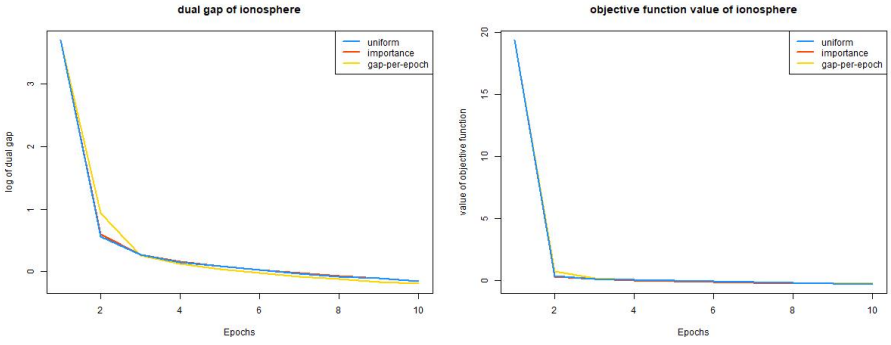
```r
svm.gap <- function(alpha,lambda = 0.1,A,dimension,y,w){
  # 计算对偶gap
  i <- dimension
  n <- dim(A)[2]
  gap <- positive(1-y[i]*sum(A[,i]*w))/n - alpha[i]*y[i]/n +
alpha[i]*sum(A[,i]*w)
  if(gap<0){
    return(0)
  }
  return(gap)
}
```

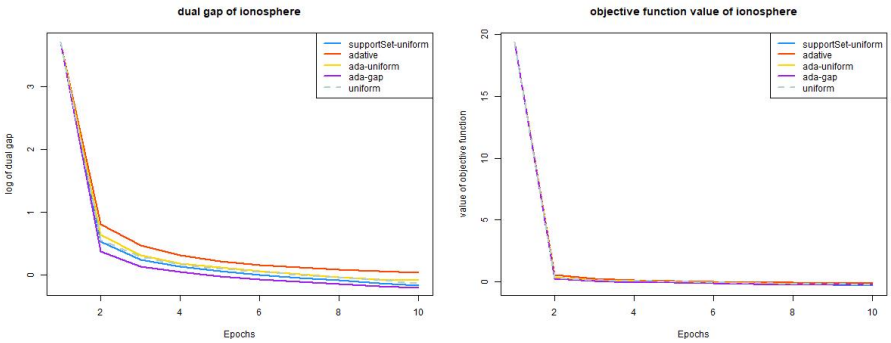
```r
svm.loss <- function(A,alpha,lambda = 0.1,y) {
  # 返回loss函数的值
  n <- dim(A)[2]
  loss <- (norm(as.matrix(A%*%alpha ),type = "2"))^2/(n^2 * lambda *
2)-sum(alpha*y)/n
  return (loss)
}
```

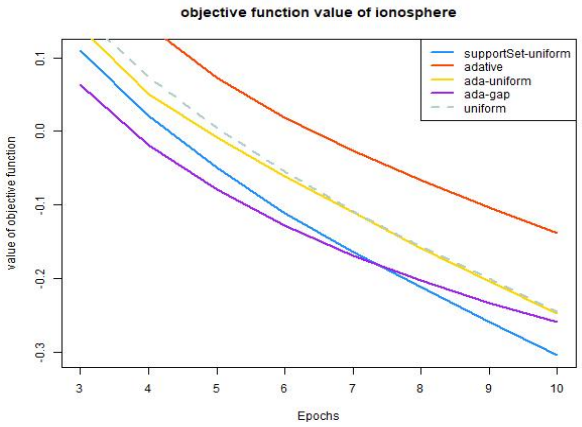
```
svm.dualres <- function(A,dimension,y,w,alpha){
  #计算对偶残差
  i <- dimension
  n <- dim(A)[2]
  flag <-  0
  eps <- 1e-5
  input <- -t(A[,i])%*%w
  if (y[i]==1){
    if (input <= -1 - eps){
    g_sub <- 0
   }else if(input >= -1 + eps){
     g_sub <- 1/n
   }else{
    g_sub_right <- 1/n
    g_sub_left  <- 0
    flag <- 1 }
  }else{
    if (input <= 1 - eps){
    g_sub <- -1/n
   }else if(input >= 1 + eps){
     g_sub <- 0
   }else{
    g_sub_right <- 0
    g_sub_left  <- -1/n
    flag <- 1 }
  }
```

```
  if (flag == 0){
      return (abs(alpha[i]-g_sub))
  }
  if (alpha[i] >= g_sub_left & alpha[i] <= g_sub_right){
    return (0)
  }
  else {
    return (min(abs(alpha[i]-g_sub_right),abs(alpha[i]-g_sub_left)
))
  }
}
```

4.3.3 Result for SVM in ionosphere







The result obtained is somewhat different from the original one. This result is actually obtained by substituting zero when the dual gap has a negative value (this issue will be discussed later). As for suboptimality, the article does not clearly indicate the definition of the SVM problem. So what we draw is the change curve of the value of the objective function.

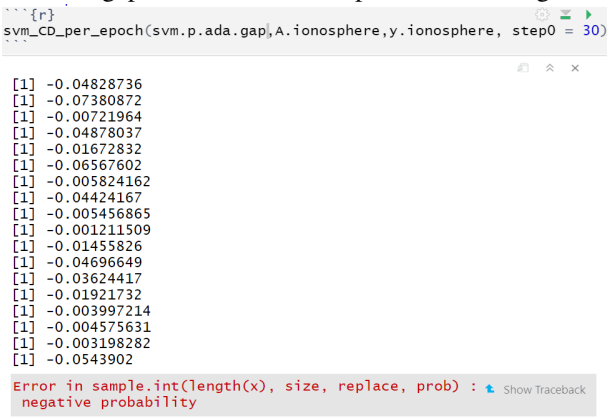
We feel that there is a certain problem in the processing of svm in the article, otherwise there will be no negative dual gap (will be explained in detail later), but we are not very clear about the specific principle of svm, so we do not know where the problem of svm is handled in the article . Therefore, we won't discuss the results too much here, just show them.

4.3.4 Problems in SVM

The article said that the dual gap is always non-negative. So in our code, when the gap is negative, we set it to be zero. That's because we thought that some sort of calculation error may lead the dual gap that should be zero to be negative.

> While always non-negative, under strong duality the gap reaches zero only in an optimal pair $(\boldsymbol{\alpha}^\star, \boldsymbol{w}^\star)$. When f is differentiable the optimality conditions (2) write as $\boldsymbol{w}^\star = \boldsymbol{w}(\boldsymbol{\alpha}^\star) = \nabla f(A\boldsymbol{\alpha}^\star)$. We will rely on the following relationship in our applications:
>
> $$\boldsymbol{w} = \boldsymbol{w}(\boldsymbol{\alpha}) := \nabla f(A\boldsymbol{\alpha}) \ .$$

When it is prompted that a negative probability vector appears when the sample_p function is called, we modify the svm_gap function so that it prints out the negative Dual Gap and get the result as shown in the figure below:



However, all the expressions were checked and found to have been typed correctly. So we assume that this part of the article may have little mistakes.

5 Conclusion

In this project, we reproduce the adaptive coordinate descent method which adopt non-uniform sampling strategy. Respectively using the dual gap or dual residual as the value of the loss of each coordinate, we implement two basic part algorithms and derive some mixed algorithms. Finally we test our algorithms on the real data and obtain most satisfactory results and find some errors in the original paper.