IMPLEMENTATION OF COORDINATE DESCENT ALGORITHMS FOR LASSO LOGISTIC REGRESSION

Xiaosu Tong

tongx@purdue.edu

1. Introduction

Tibshirani (1996) introduced an interesting method for shrinkage and variable selection in linear models, called "Least Absolute Shrinkage and Selection Operator (LASSO)".[1] It achieves better prediction accuracy by shrinkage as the ridge regression does, but at the same time, it gives a sparse solution, which means that some regression coefficients are exactly zero. Hence, LASSO can be thought to achieve the shrinkage and variable selection simultaneously.

Actually, the LASSO is one kind of penalized regression which applies a l_1 penalization that, as opposed to ridge regression, gives rise to sparse models, ruling out the influence of most of the covariates on the response. [2] The main idea of LASSO is to use the l_1 constraint in the regularization step. That is, the estimator is obtained by minimizing the empirical risk subject to that the l_1 norm of the regression coefficients is bounded by a given positive numbers.

One important issue in generalized LASSO is that the objective function is not differentiable around zero due to the l_1 constraint, and hence special optimization techniques are needed. In this report, I propose a coordinate gradient descent algorithm which is based on traditional CLG (Combined local and global) algorithm[3].

The paper is organized as follow. In Section 2, I define the problem of LASSO logistic regression and give all the formulas that will be used in the algorithm. In Section 3, I propose the coordinate gradient descent algorithm based on CLG. In Section 4, I implement the algorithm and use simulation to evaluate the performance by demonstrating it on "a1a.txt" data file. I also summarize the result of my all results.

2. Logistic Regression with Regularization

Suppose that $y_1, ..., y_n$ are independent binary response variables that take a value of -1 or 1, and that $X_1, ..., X_n$ are corresponding covariates with $X_1 = (1, x_{i1}, ..., x_{ip})$, $1 \le i \le n$. Then the logistic regression is a conditional probability model of the form:

$$P(y_i = 1 | \beta, X_i) = \frac{1}{1 + \exp(-\beta^T X_i)} \tag{1}$$

Maximum likelihood estimation of the parameters β is equivalent to minimizing the negated log-likelihood:

$$l(\beta) = \sum_{i=1}^{n} \ln(1+\exp(-y_i \beta^T X_i)) \tag{2}$$

Correspondingly, finding the LASSO logistic regression requires minimization of:

$$J(\beta \mid \lambda) = \frac{1}{n}\sum_{i=1}^{n} \ln(1+\exp(-y_i \beta^T X_i)) + \lambda \sum_{j=1}^{d} \left|\beta_j\right| \tag{3}$$

where λ is a common nonnegative penalty parameter. Minimization of this objective function makes use of the derivative. We refer to the gradient of $L(\beta)$ as the score vector whose components are defined by:

$$s_j(\beta) = \frac{\partial l(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \frac{x_{ij} y_i}{1+\exp(\beta^T X_i y_i)} \tag{4}$$

3. Algorithm

The choice of a proper algorithm to solve the minimization of (3) is a main issue, as it needs to deal with the problem of non-differentiability of the absolute value function around zero. Furthermore, efficiency of the algorithm is fundamental, given the high dimension of the problems at hand.

Of these, I base my work on the cyclic coordinate descent algorithm [4] due to its speed and simplicity of implementation. This algorithm requires an objective function that is convex and smooth, namely having a continuous first order derivative and a piece-wise continuous second order derivative. Obviously this does not hold for our objective function (3), which is still convex, but does not have a finite derivative at 0. So modifications are needed to adapt the algorithm for the LASSO logistic regression.

As the name "cyclic coordinate descent" suggests, each step of the algorithm approximately minimizes the objective along one coordinate before going on the next, in a fashion that guarantees the overall optimum is approached. The way it's done guarantees the convergence and speed of the algorithm. [5] The idea is to construct an upper bound on the second derivative of the objective on an interval around the current value; since the objective is convex, this will give rise to the quadratic upper bound on the objective itself on that interval. Minimizing the quadratic bound on interval gives one step of the algorithm.

Let $f(\beta_j)$ be the objective expressed as the function of only one component β_j with all the rest being fixed, and $Q(\beta_j, \Delta_j)$ be an upper bound on the second derivative of $f(\cdot)$ in the Δ_j-vicinity of β_j, $\Delta_j > 0$:

$$Q(\beta_j, \Delta_j) \geq f''(\beta_j + \delta) \text{ for all } \delta \in \left[-\Delta_j, \Delta_j\right] \tag{5}$$

The minimum of the quadratic upper bound on f will then be located at $\beta_j + \Delta\upsilon_j$ where

$$\Delta\upsilon_j = -f'(\beta) / Q(\beta, \Delta_j) \tag{6}$$

This is a tentative step, we now need to check if $\Delta\upsilon_j$ happens to fall inside the interval $\left[-\Delta_j, \Delta_j\right]$; if not, the step is reduced to the interval boundary.

For the objective in (3) the upper bound takes the form:

$$Q(\beta_j, \Delta_j) = \sum_{i=1}^{n} x_{ij}^2 F(\beta^T X_i y_i, \Delta_j x_{ij}) \tag{7}$$

where the function $F(r,\delta)$ is the tight upper bound[5]:

$$F(r,\delta) = \begin{cases} 0.25, & if\ |r| \le |\delta| \\ 1/(2+\exp(|r|-|\delta|)+\exp(|\delta|-|r|)), & otherwise. \end{cases} \tag{8}$$

The objective (3) is convex and satisfies the above formulated smoothness requirements everywhere except at 0. So the tentative step computation takes the form:

$$\Delta\upsilon_j = \frac{\sum_{i=1}^{n} \frac{x_{ij} y_j}{1+\exp(\beta^T X_i y_i)} - \lambda s}{\sum_{i=1}^{n} x_{ij}^2 F(\beta^T X_i y_i, \Delta_j x_{ij})} \tag{9}$$

where $s = \frac{\beta_j}{|\beta_j|}$, and $\beta_j \neq 0$. At each tentative step we have to check if this step is trying to cross over 0, and in that case reduce it to 0. In case when the current value of β_j is 0 we have to try both directions to see if either of them gives an improvement to the objective (could not be both due to the convexity[5]). If neither works then β_j stays at 0 for the iteration.

The algorithm will be showed as below:

```
(1)    Initialize $\beta_j \leftarrow 0$, $\Delta_j \leftarrow 1$ for $j = 1,...,d$
(2)    **for** $k = 1,2,...$ **until** convergence
(3)        **for** $j = 1,2,...,d$
(4)            **if** $\beta_j = 0$
(5)                $s \leftarrow 1$   (try positive direction)
(6)                compute $\Delta\upsilon_j$ by formula (9)
(7)                **if** $\Delta\upsilon_j \le 0$ (positive direction failed)
(8)                    $s \leftarrow -1$ (try negative direction)
```

```
(9)                 compute $\Delta\upsilon_j$ by formula (9)
(10)                **if** $\Delta\upsilon_j \le 0$ (negative direction failed)
(11)                    $\Delta\upsilon_j \leftarrow 0$
(12)                **end if**
(13)            **end if**
(14)        **else**
(15)            $s \leftarrow \beta_j / |\beta_j|$
(16)            compute $\Delta\upsilon_j$ by formula (9)
(17)            **if** $s \times (\beta_j + \Delta_j) < 0$ (cross over zero)
(18)                $\Delta\upsilon_j \leftarrow -\beta_j$
(19)            **end if**
(20)        **end if**
(21)        $\Delta\beta_j \leftarrow \min(\max(\Delta\upsilon_j, -\Delta_j), \Delta_j)$ (reduce the step to the interval)
(22)        $\beta_j \leftarrow \beta_j + \Delta\beta_j$
(23)        $\Delta_j \leftarrow \max(2|\Delta\beta_j|, \Delta_j / 2)$
(24)    **end if**
(25) **end if**
```

4. Result

To study whether LASSO logistic regression provides an efficient and effective approach, I tested it on the "a1a.txt" data file. Since the dataset does include the intercept, I add a new column with all value 1 in X matrix, which will help us to find estimate value of intercept. In the data, totally we have 1605 observations.

4.1 Train parameters with single lambda

In this section I will show the number of iterations until convergence and the final penalized log-likelihood (3) for 11 different λ values which $\lambda \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$, and also the precision that equal to proportion between the number of right estimation points and the total number of observations.

Lambda	Objective	Iteration time	Precision
0	0.297919	104	86.23%
0.01	0.432581	60	82.18%
0.02	0.475608	49	81.25%
0.05	0.534710	20	75.39%
0.1	0.558016	7	75.39%
0.2	0.558016	5	75.39%
0.5	0.558016	5	75.39%
1	0.558016	5	75.39%
2	0.558016	5	75.39%
5	0.558016	5	75.39%
10	0.558016	5	75.39%

From the output, we can easily see that with the λ increasing, the final value of objective function will be increased until the original value of log-likelihood function. This is because after λ is larger than 0.1, all the β_j will be held at zero point except the intercept and β_0 is not been included in the penalty term. The iteration time is basically decreasing and the precision is decreasing a lot, especially when λ is larger than 0.1. Those results can be explained by the meaning of the λ. We know that the basic requirement for the weight λ is that its value should be large enough to get $\hat{\beta}_j = 0$ if the true value β_j is zero, and small otherwise. So if the λ is extremely large, the algorithm will kick all the independent variables out of our model.

4.2 Warm start

In this section I will solve the problem for $\lambda = 0$, and use this solution as the initial guess for β for $\lambda = 0.01$. Use the solution for $\lambda = 0.01$ to initialize the search for $\lambda = 0.02$, and so on.

Lambda	Objective	Iteration time	Precision
0	0.297917	104	86.23%
0.01	0.432580	82	82.18%
0.02	0.475604	34	81.25%
0.05	0.534710	18	75.39%
0.1	0.558016	12	75.39%
0.2	0.558016	1	75.39%
0.5	0.558016	1	75.39%
1	0.558016	1	75.39%
2	0.558016	1	75.39%
5	0.558016	1	75.39%
10	0.558016	1	75.39%

From the output, we can see that there is no too much difference between the result from the 4.1 and 4.2. The final objective function values are more or less the same for each lambda, as well as the precision. However, when we focus on the iteration times, we can find that for large lambda, the iteration time becomes smaller, and the total iteration time become smaller too, which means the warm start method does speed up our convergence time.

4.3 Sparse data

If we have a sparse dataset, it is better to use link list data structure to store our data. In my data structure for whole dataset, I include a one dimension array to store the y value and n link lists to store the X matrix, where n is the number of observations. Once we use the link list as our data structure, it is very convenient to implement vector inner production, as well as the non-zero element selection. Suppose our independent variable is d dimension with $m \in o(d)$ non-zero elements, and data has n observations. For instance, during the calculation of formula (9), link list will shrink the time complexity from $O(nd^2)$ to $o(nd^2)$. The algorithm in this section will be exactly the same as the one we discussed in section 3.

4.4 Lambda selection

Selection of the tuning constant λ can be guided by cross-validation. Recall that in k-fold cross-validation, one divides the data into k equal batches (subsamples) and estimates parameters k times, leaving one batch out per time. The testing error for each omitted batch is computed using the estimates derived from the remaining batches. Based on the "a1a.txt" dataset, I calculate the precision for each evaluation batch for each λ value, and then use comparison t-test to analyze which λ value will give us the best prediction.

Letter name	Batch1	Batch2	Batch3	Batch4	Batch5
0	79%	77%	78%	85%	80%
0.01	79%	74%	76%	79%	82%
0.02	79%	74%	76%	75%	76%
0.05	79%	74%	76%	75%	75%
0.1	79%	74%	76%	75%	72%
0.2	79%	74%	76%	75%	72%
0.5	79%	74%	76%	75%	72%
1	79%	74%	76%	75%	72%
2	79%	74%	76%	75%	72%
5	79%	74%	76%	75%	72%
10	79%	74%	76%	75%	72%

From the output, we can see that when λ is larger than 0.1, there is no different between different λ value with respect to the precision, so we only have to compare 0, 0.01, 0.02, and 0.05.

```
          Variable    N          Mean        Std Dev
-----------------------------------------------------
          g1          5    79.8000000      3.1144823
          g2          5    78.0000000      3.0822070
          g3          5    76.0000000      1.8708287
          g4          5    75.8000000      1.9235384
          g5          5    75.2000000      2.5884358
-----------------------------------------------------
```

According to the summary above, we calculate the t-score between each two groups. Our hypotheses are: $H_0 : \mu_i > \mu_j$ vs $H_a : \mu_i \le \mu_j$. Then we can get our conclusion: $\mu_1 > \mu_2$, $\mu_1 > \mu_2$, $\mu_1 > \mu_3$, $\mu_1 > \mu_4$, $\mu_1 > \mu_5$, which means $\lambda = 0$ provide the best prediction for our dataset.

Reference:

[1] R. Tibshirani. Regression on shrinkage and selection via the lasso. Journal of the Royal Statistical Society (Series B), 58:267-288, 1996

[2] Fu, W. J. (1998). Penalized regressions: The bridge versus the lasso. J. Compute. Graph. Statist. 7397–416. MR1646710

[3]T. Zhang and F. Oles. Text categorization based on regularized linear classifiers. Information Retrieval, 4(1):5-31, April 2001.

[4] Wu, T. T. and Lange, K. (2008). Supplement to “Coordinate descent algorithms for lasso penalized regression.” DOI: 10.1214/07-AOAS174SUPP.

[5] A Genkin, DD Lewis, D Madigan. Sparse Logistic Regression for Text Categorization.

[6] Zhao, P. and Yu, B. (2006). On model selection consistency of LASSO. J. Machine Learning Res. 7, 2541-2567.