SCIENCE CHINA
Information Sciences

• LETTER •

June 2020, Vol. 63 169102:1–169102:3
https://doi.org/10.1007/s11432-018-9832-y

An accelerator for the logistic regression algorithm based on sampling on-demand

Jiye LIANG[1*], Yunsheng SONG[2], Deyu LI[1], Zhiqiang WANG[1] & Chuangyin DANG[3]

[1] *Key Laboratory of Computational Intelligence and Chinese Information Processing of Ministry of Education, School of Computer and Information Technology, Shanxi University, Taiyuan* 030006*, China;*
[2] *College of Information Science and Engineering, Shandong Agricultural University, Tai'an* 271018*, China;*
[3] *Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong* 999077*, China*

Received 2 November 2018/Revised 17 January 2019/Accepted 15 March 2019/Published online 14 April 2020

Citation Liang J Y, Song Y S, Li D Y, et al. An accelerator for the logistic regression algorithm based on sampling on-demand. Sci China Inf Sci, 2020, 63(6): 169102, https://doi.org/10.1007/s11432-018-9832-y

Dear editor,
Training a logistic regression classifier is equivalent to solving a convex optimization problem, which is frequently solved with gradient descent (GD). As the gradient is computed using all the data, using GD is very time-consuming when the dataset is large [1, 2]. To accelerate the GD algorithm, stochastic gradient descent (SGD) estimates the gradient using only one training data point, which can substantially reduce the amount of computation for large datasets [3]. However, SGD does not guarantee that the estimated gradient is a descent direction of the objective function, so many iterations are needed to obtain the final solution [4].

An alternative approach is mini-batch gradient descent (MGD), which estimates the gradient by sampling some of the training data. This method can reduce the variance of the estimated gradient, has more stable convergence, and is able to yield a more accurate solution to the optimization problem [5, 6]. However, choosing the sample size for different kinds of objective functions remains a difficult problem. In this study, we propose a strategy of sampling on-demand to speed up the logistic regression algorithm. The accelerator generates a random sample set by sampling iteratively until all components of the estimated gradient satisfy a specified stopping criteria. At each iteration, the accelerator only computes the components that did not satisfy the stopping criteria at the previous iteration. We prove that the estimated gradient is a descent direction for the objective function with a sufficiently large probability.

Problem formulation and optimization. Let $\mathrm{TS} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ be a training dataset, where each instance $x_i = (x_{i1}, x_{i2}, \dots, x_{im})^{\mathrm{T}}$ is a vector of m attributes, $y_i \in \{0, 1\}$ is the label for x_i, and N is the number of instances. For the given dataset TS, the main task in training a logistic regression classifier is solving the optimization problem $\min_{\omega\in\mathbb{R}^{m+1}} J(\omega)$, where $J(\omega) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i\omega^{\mathrm{T}}z_i - \ln(1 + \exp(\omega^{\mathrm{T}}z_i))\}$, $z_i = (z_{i,1}, z_{i,2}, \dots, z_{i,m+1})^{\mathrm{T}} = (1, x_{i1}, \dots, x_{im})$.

The GD algorithm is the most commonly used algorithm for solving the above optimization problem. The step-size is proportional to the negative of the gradient of the objective function at each iteration. Let ω^k be the weight vector obtained by the GD algorithm at the k-th iteration. The current gradient $\nabla J(\omega^k) = (\nabla J_1(\omega^k), \nabla J_2(\omega^k), \dots, \nabla J_{m+1}(\omega^k))$ is computed as

$$\left(\frac{1}{N}\sum_{i=1}^{N}\nabla l_1(\omega^k, z_i, y_i), \dots, \frac{1}{N}\sum_{i=1}^{N}\nabla l_{m+1}(\omega^k, z_i, y_i)\right),$$

where $\nabla l_j(\omega^k, z_i, y_i) = -\big(\frac{\exp(\omega^{k\mathrm{T}}z_i)}{1+\exp(\omega^{k\mathrm{T}}z_i)} - y_i\big)z_{i,j}$, $j = 1, 2, \dots, m + 1$. Furthermore, the solution ω^* obtained by GD is a globally optimal solution as

* Corresponding author (email: ljy@sxu.edu.cn)

© Science China Press and Springer-Verlag GmbH Germany, part of Springer Nature 2020 info.scichina.com link.springer.com

the objective function $J(\omega)$ is a convex function of ω. However, computing the gradient is very time-consuming when the dataset is large. To mitigate this deficiency, we propose an accelerated logistic regression algorithm based on sampling on-demand (LR-SOD).

Our approach. The LR-SOD algorithm is a type of MGD algorithm. At each iteration, it first randomly generates a subset of the dataset TS, and then obtains an estimate $\nabla\widehat{J}(\omega)$ of the gradient $\nabla J(\omega)$ from this subset and updates the current weight vector ω using $\nabla\widehat{J}(\omega)$ instead of $\nabla J(\omega)$. We note that the number of iterations required to achieve convergence is closely related to the estimate $\nabla\widehat{J}(\omega)$. If $-\nabla\widehat{J}(\omega)$ does not ensure a sufficient descent in $J(\omega)$ at every iteration, then the number of iterations can be large. Therefore, we provide a rule for obtaining a good estimate $\nabla\widehat{J}(\omega)$. Naturally, we would like $\nabla\widehat{J}(\omega)$ to be close to $\nabla J(\omega)$ in some sense. Thus we use the following inequality:

$$\nabla\widehat{J}(\omega)^{\mathrm{T}}\nabla J(\omega) \geqslant (1-\varepsilon)(\|\nabla J(\omega)\|_2)^2, \qquad (1)$$

where $\varepsilon \in (0, 1/2)$, and $\|\nabla J(\omega)\|_2$ is the 2-norm of the vector $\nabla J(\omega)$. It is easily proved that the vector $\nabla\widehat{J}(\omega)$ is a descent direction for the function $J(\omega)$, as long as it satisfies inequality (1), according to [7]. Moreover, the objective function $\widehat{J}(\omega)$ is convex, so the LR-SOD algorithm is able to obtain the globally optimal solution.

To find a vector $\nabla\widehat{J}(\omega)$ that satisfies inequality (1), we propose a new sampling algorithm. For convenience of expression, each element $(x_i, y_i) \in$ TS is enlarged to (z_i, y_i) for $i = 1, 2, \ldots, N$, from which we obtain the new set $T = \{(z_1, y_1), (z_2, y_2), \ldots, (z_N, y_N)\}$. Although simple random sampling is the simplest way to achieve a sample, it requires a pre-determined sample size. Instead of pre-determining the sample size, our algorithm samples points sequentially from T into S until the value of a function defined on S meets a stopping condition. The stopping condition plays an important role in this adaptive sampling. In the following, we introduce a process for constructing a stopping condition that satisfies inequality (1).

The vector $\nabla J(\omega; S)$ is an estimate of the gradient $\nabla J(\omega)$, and each component $\nabla J_j(\omega; S) = (1/|S|)\sum_{(z_i,y_i)\in S} l_j(\omega, z_i, y_i)$ is an estimate of $\nabla J_j(\omega)$, $j = 1, 2, \ldots, m+1$. Thus determining the size of the subset S to guarantee that $\nabla J(\omega; S)$ satisfies the inequality (1) is a multidimensional estimation problem. To deal with this problem, we first divide the multidimensional problem into $m+1$ one-dimensional subproblems and solve each subproblem. Using the definitions of the inner product and the norm, inequality (1) holds if $\sum_{j=1}^{m+1}\nabla J_j(\omega; S)\nabla J_j(\omega) \geqslant \sum_{j=1}^{m+1}(1-\varepsilon)(\nabla J_j(\omega))^2$. This means that inequality (1) always holds when $J_j(\omega; S)$ and $J_j(\omega)$ satisfy the inequality

$$\nabla J_j(\omega; S)\nabla J_j(\omega) \geqslant (1-\varepsilon)(\nabla J_j(\omega))^2 \qquad (2)$$

for all $j = 1, 2, \ldots, m+1$. That is, the problem of obtaining a vector that satisfies the inequality (1) can be transformed into the problem of finding $m+1$ components $\nabla J_j(\omega; S)$ that satisfy inequality (2). Because each component $\nabla J_j(\omega)$ of the vector $\nabla J(\omega)$ is unknown before sampling, inequality (2) cannot be used as a stopping criteria in our method. However, the value of $|\nabla J_j(\omega)|$ is fixed for a given weight vector ω, and can be approximated by a strictly decreasing function $\alpha_j^t = d_j\sqrt{\ln((m+1)ts(ts+s)/\delta)/(2st)}$ in the number of samples t, where s is a fixed integer, $\delta \in (0, 0.5)$, and $d_j = \max\{b_j - a_j, -2a_j, 2b_j\}$, $a_j = \min_{1\leqslant i\leqslant N} z_{i,j}$, $b_j = \max_{1\leqslant i\leqslant N} z_{i,j}$ for $j = 1, 2, \ldots, m+1$. As the LR-SOD algorithm requires multiple iterations to sample from T into S, repeatedly computing $\nabla J(\omega; S)$ and checking whether each component satisfies the stopping condition can take a long time. To address this issue, we make three improvements. First, we sample in batches instead of one at a time. For example, we sample s points in each iteration rather than one point, where s is a positive integer. Moreover, the change in α_j^t is proportional to the value of s according to the definition. Therefore, this operation also causes the components of $\nabla J(\omega; S)$ to reach their stopping condition more quickly. Second, we compute $\nabla J(\omega; \widetilde{S}_t) = 1/|\widetilde{S}_t|(|\widetilde{S}_{t-1}|\nabla J(\omega; \widetilde{S}_{t-1}) + \sum_{(z_i,y_i)\in S_t} l(\omega, z_i, y_i))$ for the subsets $\widetilde{S}_{t-1}$ and S_t, where $\widetilde{S}_{t-1}$ is the points sampled in the first $t-1$ iterations, S_t is the points sampled in the t-th iteration, and $\widetilde{S}_t = S_t \cup \widetilde{S}_{t-1}$. Finally, each component of the vector $\nabla\widehat{J}(\omega)$ can be obtained in a different iteration. If the component $\nabla J_j(\omega; \widetilde{S}_t)$ reaches its stopping condition in the t_0-th iteration, then it is accepted as the j-th component of $\nabla\widehat{J}(\omega)$ ($\nabla\widehat{J}_j(\omega) = \nabla J_j(\omega; \widetilde{S}_{t_0})$), and will not be updated in any subsequent iterations. Theorem 1 shows that the gradient estimate $\nabla\widehat{J}(\omega)$ obtained using the LR-SOD algorithm is a descent direction for the current objective function with large probability.

Theorem 1. For $0 < \varepsilon < 1/2$ and $0 < \delta < 1/2$, the LR-SOD algorithm outputs a vector $\nabla\widehat{J}(\omega)$ such that $P(\nabla\widehat{J}(\omega)^{\mathrm{T}}\nabla J(\omega) \geqslant (1-\varepsilon)\|\nabla J(\omega)\|^2)$ with probability $1-\delta$.

Experiments. To demonstrate the efficiency of our proposed algorithm, three representative GD

Table 1 The execution time (s) for four algorithms

Dataset	LR-GD	LR-DSG	LR-BGD	LR-SOD
Cifa	8586.47	114.65	119.37	78.53
Cod-rna	4827.57	133.21	73.33	43.39
Covtype	7948.21	115.89	285.77	29.34
Ijcnn1	1900.81	31.71	80.62	29.64
Mnist	5272.12	68.29	376.22	63.23
Rcv1	3370854.18	7296.56	4050.76	1298.32
Real-sim	243920.00	1139.20	761.60	320.00
Skin-nonskin	3321.46	203.09	199.81	163.78
Susy	67645.77	757.83	11286.56	735.76

algorithms were selected for comparison, namely GD with all the data (LR-GD), GD with static samples (LR-BGD) and the dynamic sample gradient algorithm (LR-DSG).

As shown in Table 1, the LR-SOD algorithm requires less training time than LR-DSG and LR-BDG. As opposed to the LR-SOD algorithm, the sample size for estimating the gradient may increase with each iteration in the LR-DSG algorithm, so that LR-DSG requires a lot of time to compute the estimated gradient. Furthermore, both LR-DSG and LR-BGD determine the sample size before sampling, and they sample from the training data set using this fixed sample size to obtain the estimated gradient. Because different data points have different contributions to the gradient, the gradients estimated by these two algorithms is often not a descent direction for the current objective function. Therefore, these two algorithms require more iterations to obtain a final solution, which also increases the execution time. The LR-SOD algorithm guarantees that the estimated gradient is a descent direction for the current objective function at each iteration with large probability, so it requires a much smaller number of iterations to obtain a final solution.

Conclusion. To effectively deal with large-scale data, this study presents a new sampling on-demand logistic regression algorithm. This algorithm generates a random sample by successively sampling data points until all the components of the estimated gradient satisfy a specified stopping condition. At each iteration, only the components that do not satisfy the stopping condition are estimated, while the components that already satisfy their stopping condition become the corresponding components of the final gradient estimate. We also prove that the estimated gradient using LR-SOD is a descent direction for the current objective function with large probability. The experimental results on several large-scale datasets further confirm the effectiveness and efficiency of LR-SOD.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61876103, 61432011), Project of Key Research and Development Plan of Shanxi Province (Grant No. 201603D111-014), and 1331 Engineering Project of Shanxi Province.

References

1 Bai G T, Tao R, Zhao J, et al. Fast FOCUSS method based on bi-conjugate gradient and its application to space-time clutter spectrum estimation. Sci China Inf Sci, 2017, 60: 082302

2 Li C H, Zhang E C, Jiu L, et al. Optimal control on special Euclidean group via natural gradient algorithm. Sci China Inf Sci, 2016, 59: 112203

3 Nesterov Y. Primal-dual subgradient methods for convex problems. Math Program, 2009, 120: 221–259

4 Lei L H, Jordan M I. Less than a single pass: stochastically controlled stochastic gradient. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Florida, 2017. 148–156

5 Wright S J. Accelerated block-coordinate relaxation for regularized optimization. SIAM J Optim, 2012, 22: 159–186

6 Shalev-Shwartz S, Zhang T. Stochastic dual coordinate ascent methods for regularized loss minimization. J Mach Learn Res, 2013, 14: 567–599

7 Lubin M, Dunning I. Computing in operations research using Julia. INFORMS J Comput, 2015, 27: 238–248