ALLN 2006 L_1 Regularized Logistic Regression

Vasco Villas-Boas

September 5, 2022

L_1 Regularized Logistic Regression

Logistic regression is a technique used to classify data into two groups and it can be generalized into classifying into many groups with multinomial logistic regression. In our tennis setting, we only care about two groups- Player A wins the match or Player B wins the match. Mathematically, suppose we're given data $D := (x_i, y_i)_{i=1}^N$ where each x_i is a vector of k characteristics for data point i, $y_i \in \{-1, 1\}$ is a classification for data point i, and N is the number of data points. In the logistic model, given a k dimensional parameter vector β, we have a function:

$$\Pr(y = 1|x;\beta) := \sigma(\beta^T x) := \frac{1}{1 + e^{-\beta^T x}} \tag{1}$$

This function $\Pr(y = 1|x;\beta)$ aims to represent the probability that the set of characteristics x have class label $y = 1$. The regression part of logistic regression specifies how we want to pick our parameter β.

For the sake of L_1 Regularization, suppose we impose a zero-mean Laplacian Prior on β: $\Pr(\beta; b > 0) = (\frac{b}{2})^k e^{-b||\beta||_1}$. We should interpret this prior as favoring the different components of β near 0. Further, it favors sparsity over L_2 distance from 0. For example, in the $k = 2$ case, if $||\beta||_1 = A$, then it favors $\beta_1 = [5, 0]$ over $\beta_2 = [3, 3]$, even though $||\beta_1||_2 > ||\beta_2||_2$.

Now, if we want to compute the compute the Maximum-A-Priori (*MAP*) estimate of β given our data, logistic model, and the assumption that all of our data points are independent:

$$\begin{aligned}
MAP(\beta|D;b) &= \text{argmax}_\beta \Pr(\beta|D) \\
&= \text{argmax}_\beta \Pr(D|\beta)\Pr(\beta) \\
&= \text{argmax}_\beta \Pr(\beta)\Pi_{i=1}^N \Pr(y = y_i|x_i;\beta) \\
&= \text{argmin}_\beta - \log((\frac{b}{2})^k) - \log(e^{-b||\beta||_1}) - \sum_{i=1}^N \Pr(y = y_i|x_i;\beta) \\
&= \text{argmin}_\beta b||\beta||_1 - \sum_{i=1}^N \Pr(y = y_i|x_i;\beta) \qquad (2) \\
&= \text{argmin}_\beta - \sum_{i=1}^N \Pr(y = y_i|x_i;\beta) \text{ st } ||\beta||_1 \le c \qquad (3)
\end{aligned}$$

The last two minimization problems give the L_1 Regularized Logistic Regression Problem in two different equivalent forms. The two problems are equivalent because for any given value of b in the first problem, there exists a value of c in the second problem such that the two objective functions are equal (Rockefellar 1970); the opposite is also true. As a result, the minimizing β is the same in both problems. As a first remark, b and/ or c are typically picked via cross-validation. As a second remark, the un-regularized logistic regression problem is

$$MLE(\beta|D;b) = \text{argmin}_\beta - \sum_{i=1}^N \Pr(y = y_i|x_i;\beta) \tag{4}$$

The derivation is quite similar except there's no prior on β since we're in that case computing $MLE(\beta|D)$.

ALLN 2006 Algorithm for L_1 Regularized Logistic Regression

In this subsection, I aim to summarize the L_1 Regularized Logistic Regression Algorithm of Su-In Lee, Honglak Lee, Pieter Abbeel and Andrew Y. Ng (2006), from now on shortened to ALLN (2006), and dive deeper into some technical points to make the algorithm more accessible. The objective generally solving is to iteratively solve a L_1 regularized logistic regression problem.

Newton-Method- 2nd Order Approximation

For simplicity, to start, suppose we're trying to iteratively solve the unregularized logistic regression problem given in equation 4. For notational reasons, let's call the objective of equation 4: $g(\beta) := -\sum_{i=1}^{N} \Pr(y = y_i|x_i; \beta)$. At iteration j, we have our current parameter $\beta^{(j)}$ and we think about how we want to reach $\beta^{(j+1)}$ to get closer to our optimal β. Recall from calculus that we can expand using a Taylor expansion:

$$g(\beta^{(j)} + \Delta\beta) = g(\beta^{(j)}) + \nabla g(\beta^{(j)})\Delta\beta + \frac{1}{2}\Delta\beta^T \nabla^2 g(\beta^{(j)})\Delta\beta + ...$$

If we truncate this Taylor expansion at the second order term, and we replace $H(\beta^{(j)}) := \nabla^2 g(\beta^{(j)})$, where H is the hessian matrix at $\beta^{(j)}$:

$$g(\beta^{(j)} + \Delta\beta) \approx g(\beta^{(j)}) + \nabla g(\beta^{(j)})\Delta\beta + \frac{1}{2}\Delta\beta^T H(\beta^{(j)})\Delta\beta$$

Our objective is to find $\Delta\beta$ so as to minimize $g(\beta^{(j)} + \Delta\beta)$. Thus, we take the gradient of $g(\beta^{(j)} + \Delta\beta)$ with respect to $\Delta\beta$, and set it equal to 0:

$$\begin{aligned}
0 &= \nabla_{\Delta\beta}(g(\beta^{(j)} + \Delta\beta)) \\
&\approx \nabla_{\Delta\beta}(g(\beta^{(j)}) + \nabla g(\beta^{(j)})\Delta\beta + \frac{1}{2}\Delta\beta^T H(\beta^{(j)})\Delta\beta) \\
&= \nabla g(\beta^{(j)}) + H(\beta^{(j)})\Delta\beta
\end{aligned}$$

Solving the last line for $\Delta\beta$ gives $\Delta\beta = -H^{-1}(\beta^{(j)})\nabla g(\beta^{(j)})$. We could immediately set $\beta^{(j+1)}$ equal to $\beta^{(j)} + \Delta\beta$ but recall that $\Delta\beta$ came from an approximation and so we might be able to do better and we also want to avoid divergence. Thus, like in the algorithm, let's define a temporary variable $\gamma^{(j)}$

$$\gamma^{(j)} := \beta^{(j)} + \Delta\beta = \beta^{(j)} - H^{-1}(\beta^{(j)})\nabla g(\beta^{(j)}) \tag{5}$$

It indicates our step direction. Then, we'll define $\beta^{(j+1)}$

$$\beta^{(j+1)} := \beta^{(j)} + t\gamma^{(j)} \tag{6}$$

The parameter $t \geq 0$ is found by a line search (more on that later)[1]. We have just derived the parameter update for the quadratic Newton-Method for finding the minimum of $g(\cdot)$.

Alternative Approach to Find $\gamma^{(j)}$

In practice, inverting $H(\beta^{(j)})$ is very expensive and thus sometimes researchers solve the linear equation $-\nabla g(\beta^{(j)}) = H(\beta^{(j)})\Delta\beta$ or employ some factorization on $H(\beta^{(j)})$ to find $\gamma^{(j)}$ (Hauser 2012). In other cases, Quasi-Newton Methods are preferred to not grapple with the Hessian matrix.

In the paper, ALLN documented another way to compute $\gamma^{(j)}$ without involving the Hessian. First, define the define matrix $X \in \mathbb{R}^{kxN}$:

[1] I'll explain shortly what is Back-tracking line search. Note that you could also find a "good" t by creating some feasible interval for t (ie., $[0, U]$) and doing a brute force search in $t \in [0, U]$ and pick the t that makes $g(\beta^{(j+1)})$ the smallest.

$$X := [x_1 x_2 ... x_N]$$

Next, for $i \in \{1, ..., N\}$, and for the jth iteration, define the diagonal matrix $\Lambda \in \mathbb{R}^{NxN}$ and vector $z \in \mathbb{R}^N$ as:

$$\Lambda_{i,i} := \sigma(\beta^{(j)T} x_i)(1 - \sigma(\beta^{(j)T} x_i)) \tag{7}$$

$$z_i := x_i^T \beta^{(j)} + \frac{(1 - \sigma(y_i \beta^{(j)T} x_i))y_i}{\Lambda_{i,i}} \tag{8}$$

As a result, $\nabla g(\beta^{(j)}) = X\Lambda(z - X^T \beta^{(j)})$ and $H(\beta^{(j)}) = -X\Lambda X^T$. To see these facts, first see that we can rewrite the objective of the logistic regression problem in equation 4 since $y_i \in \{1, -1\}$:

$$g(\beta) = -\sum_{i=1}^{N} \text{Pr}(y = y_i | x_i; \beta) = -\sum_{i=1}^{N} \sigma(y_i \beta^T x_i)$$

Then,

$$\begin{aligned}
\nabla g(\beta^{(j)}) &= \sum_{i=1}^{N} (1 - \sigma(y_i \beta^{(j)T} x_i)) y_i x_i \\
&= \sum_{i=1}^{N} x_i \Lambda_{i,i} \frac{(1 - \sigma(y_i \beta^{(j)T} x_i))y_i}{\Lambda_{i,i}} \\
&= -X\Lambda X^T \beta^{(j)} + \sum_{i=1}^{N} x_i \Lambda_{i,i} (x_i^T \beta_{(j)} + \frac{(1 - \sigma(y_i \beta^{(j)T} x_i))y_i}{\Lambda_{i,i}}) \\
&= -X\Lambda X^T \beta^{(j)} + \sum_{i=1}^{N} x_i \Lambda_{i,i} z_i \\
&= X\Lambda(z - X^T \beta^{(j)}) \\
H(\beta^{(j)}) &= -\sum_{i=1}^{N} \sigma(y_i \beta^{(j)T} x_i)(1 - \sigma(y_i \beta^{(j)T} x_i)) \cancelto{1}{y_i y_i} x_i x_i^T \\
&= -\sum_{i=1}^{N} x_i \sigma(\beta^{(j)T} x_i)(1 - \sigma(\beta^{(j)T} x_i)) x_i^T \\
&= -\sum_{i=1}^{N} x_i \Lambda_{i,i} x_i^T \\
&= -X\Lambda X^T
\end{aligned}$$

As a result of these facts, we can derive an expression for $\gamma^{(j)}$ from equation 5, the update direction for $\beta^{(j)}$, in terms of X, Λ, and z.

$$\begin{aligned}
\gamma^{(j)} &= \beta^{(j)} - H^{-1}(\beta^{(j)}) \nabla g(\beta^{(j)}) \\
&= \beta^{(j)} + (X\Lambda X^T)^{-1} X\Lambda(z - X^T \beta^{(j)}) \\
&= \beta^{(j)} + (X\Lambda X^T)^{-1} X\Lambda z - (X\Lambda X^T)^{-1}(X\Lambda X^T)\beta^{(j)} \\
&= (X\Lambda X^T)^{-1} X\Lambda z
\end{aligned} \tag{9}$$

If you think carefully about equation 9, $\gamma^{(j)}$ is actually also the solution to a least squares problem:

$$\gamma^{(j)} = \text{argmin}_\gamma ||\Lambda^{1/2} X^T \gamma - \Lambda^{1/2} z||_2 \tag{10}$$

We've carefully derived an approach to find the update direction $\gamma^{(j)}$ in the second order Newton Method by solving a least squares problem. There exist efficient and accurate iterative methods to approximate unregularized least squares problems and so it's a sound alternative to inverting the hessian in the standard newton method update in equation 5.

Reintroducing the Regularization

To introduce regularization to this iterative least squares approach to solve logistic regression, we augment the minimization problem in equation 10 with the L_1 constraint.

$$\gamma^{(j)} = \text{argmin}_\gamma ||\Lambda^{1/2} X^T \gamma - \Lambda^{1/2} z||_2 \quad \text{s.t. } ||\gamma||_1 \leq c \tag{11}$$

Or equivalently

$$\gamma^{(j)} = \text{argmin}_\gamma ||\Lambda^{1/2} X^T \gamma - \Lambda^{1/2} z||_2 + b||\gamma||_1 \tag{12}$$

Now, when we look for the update direction, $\gamma^{(j)}$, at iteration j, we update $\beta^{(j)}$ according to this L_1 regularized least squares problem.

The ALLN Algorithm

Algorithm 1 L_1 Regularized Logistic Regression- Iterative Lasso Least Squares

```
1: $\beta^{(0)} \leftarrow \vec{0}$
2: for j=0 to maxIterations do
3:     Compute $\Lambda$ and $z$ according to equations 7 and 8 using $\beta^{(j)}$
4:     Use a LARS regression solver to compute the solution, $\gamma^{(j)}$, to the constrained minimization problem in equation 12
5:     $\beta^{(j+1)} \leftarrow \beta^{(j)} + t\gamma^{(j)}$ where $t \geq 0$ is found using a backtracking line search that minimizes the objective of equation 2, $g(\cdot)$
6:     Evaluate the objective function of equation 2 at $\beta^{(j+1)}$
7:     if stopping condition met then
8:         return {"converged" : True, "beta" : $\beta^{(j+1)}$}
9:     end if
10: end for
11: return {"converged" : False, "beta" : $\beta^{(j+1)}$}
```

As a first remark, ALLN title their algorithm "IRLS-LARS" which stands for "Iterated Reweighted Least Squares- Least Angle Regression". Iterated Reweighted Least Squares comes from the update in equation 11: the authors choose to view that minimization problem as a solution to a weighted (weight each data point according to $\Lambda^{1/2}$) least squares problem. The least squares problem is "reweighted" since the weights $\Lambda^{1/2}$ depend on $\beta^{(j)}$ and hence change at each iteration. They use the Least Angle Regression Algorithm with the Lasso modification (Efron *et al.* 2004) to efficiently solve each the regularized least squares problem at each iteration.

Next, the algorithm above is written for one value of b, the hyperparameter used to determine the amount of regularization. The author indicates we're to use cross-validation to pick a good b.

For the stopping condition, there were two natural candidates- stop if consecutive βs were "close enough" according to some distance metric in $\mathbb{R}^k$ or stop if consecutive objective values were "close enough". Lastly, there's the notion of backtracking line search to pick the optimal t in line (5) of the algorithm that I'll discuss in the next subsection.

Backtracking Line Search

Backtracking line search is a method to determine how much to move along a given search direction to minimize a differentiable objective function with known (or estimated) gradient. It starts with a large step-size in the search direction and then gradually reduces the step-size (ie., backtracks) until the objective function decreases by an ”adequate” amount.

For the backtracking line search algorithm pseduocode, let f be the differentiable objective function we aim to iteratively minimize. Let x be the starting position, $\nabla f(x)$ be the known (or estimated) gradient of f at x, and let p be the candidate search direction[2]. Also, let $t_0 > 0$ be the maximum candidate step size, and $\tau \in (0, 1)$ and $c \in (0, 1)$ be control parameters.

Algorithm 2 Backtracking Line Search

```
1: $\alpha \leftarrow -c(\nabla f(x)^T p)$
2: $i \leftarrow 0$
3: while $f(x) - f(x + t_i p) \geq t_i \alpha - \epsilon$ do
4:     $i \leftarrow i + 1$
5:     $t_i \leftarrow \tau t_{i-1}$
6: end while
7: return $t_i$
```

The while loop stopping condition, $f(x) - f(x + t_i p) \geq t_i \alpha$ is known as the Armijo–Goldstein condition of sufficient decrease (Burke 2020). I add in the $0 < \epsilon \ll 1$ term to help with convergence of the backtracking line search algorithm. We typically pick t_0 to be fairly large since it took nontrivial effort to find the search direction p.

[2]As a remark, when conducting gradient descent, $p := \nabla f(x)$ since we search in the direction of the gradient. However, in the Newton method, using our notation above, $p := \gamma^{(j)} \neq \nabla g(x)^T$.

REFERENCES

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least Angle Regression. *The Annals of Statistics*, 32(2):407–451, 2004. URL: `http://www.jstor.org/stable/3448465`.

Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient L1 Regularized Logistic Regression. In *AAAI*, pages 401–408, 2006. URL: `http://www.aaai.org/Library/AAAI/2006/aaai06-064.php`.

R. Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970. URL: `http://www.jstor.org/stable/j.ctt14bs1ff`.