Newton's method with gradient descent for logistic regression

Thomas Tibbetts

November 2024

1 Introduction

In some situations, we have a dataset consisting of n observations and associated responses,

$$(x_i, y_i)_{i=1}^n, x_i \in \mathbb{R}^p, y_i \in \{0, 1\}$$

Each x_i is a vector of p predictor variables, while y_i is a response variable capable of taking only two values, encoded as 0 or 1 in this example. This response variable could be identified with a success/failure, true/false, or any other quantity that necessarily takes a binary value.

We often model a continuous response variable as a linear function of the predictors:

$$\hat{y_i} = \theta_0 + \theta^T x_i, \theta_0 \in \mathbb{R}, \theta \in \mathbb{R}^p$$

To adjust to the response being a binary variable, we instead attempt to model the probability that $y_i = 1$. This is accomplished using a sigmoid whose argument is a linear function of predictors:

$$\pi_i = \hat{P}(y_i = 1) := \sigma(\theta_0 + \theta^T x_i) = \frac{1}{1 + \exp\left(-(\theta_0 + \theta^T x_i)\right)}$$

The sigmoid function is used to constrain the predicted probability to the interval $(0, 1)$, but the question remains of how to determine appropriate values for θ_0 and θ. We will estimate these parameters using Maximum Likelihood Estimation (MLE).

First, some definitions are made for convenience:

$$\theta^* = \begin{pmatrix} \theta_0 \\ \theta \end{pmatrix}, \theta^* \in \mathbb{R}^{p+1}$$

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{pmatrix}, y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, \Pi = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \dots \\ \pi_n \end{pmatrix}$$

With these definitions, if we denote the ith row of X as X_i, we have that $\pi_i = {\theta^*}^T X_i \longrightarrow \Pi = {\theta^*}^T X$. For the defined problem, the negative log-likelihood function takes the form

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} [y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i)]$$

The maximum-likelihood estimator for θ^* is

$$\theta^*_{MLE} = \arg\min_{\theta} \mathcal{L}(\theta)$$

This optimization problem cannot be solved in closed-form, so iterative methods are applied, including:

- Newton's method: Beginning with an initial guess $\theta^{(0)}$, we update the approximation of θ^* using the iterative equation:

  $$\theta^{(k+1)} = \theta^{(k)} - H_{\mathcal{L}}^{-1}(\theta^{(k)}) \nabla \mathcal{L}(\theta^{(k)})$$

  The Hessian matrix of $\mathcal{L}$ can be conveniently expressed as

  $$H_{\mathcal{L}}(\theta) = X^T W X$$

where W is an $n \times n$ diagonal matrix with entries $\pi_i(1-\pi_i)$ [**1**]. Newton's method is a well-known optimization tool that often converges to the target parameter with relatively few iterations. However, its utility is limited because it often fails to converge, depending on the choice of initial guess $\theta^{(0)}$. Initial guesses outside of a close neighborhood of the minimizer may tend to cause divergence away from the optimum point.

- Gradient descent: Beginning with an initial guess $\theta^{(0)}$, we update the approximation of θ^* with the following rule:
$$\theta^{(k+1)} = \theta^{(k)} - \eta\nabla\mathcal{L}(\theta^{(k)})$$
repeatedly (with a choice of learning rate η) until some convergence condition is satisfied. The gradient of the log-likelihood can be expressed as
$$\nabla\mathcal{L}(\theta) = X^T(y - \Pi)$$
Compared to Newton's method, the method of gradient descent often takes many more iterations (and therefore a much longer runtime) to converge. However, gradient descent is advantageous in that its convergence is far less sensitive to the choice of initial guess $\theta^{(0)}$ (at least within the confines of the logistic regression problem).

- Gradient descent with momentum: This is a slight modification of gradient descent, which adds a momentum term to each update in order to smooth the trajectory of the descent [**2**]. The rule for updating the parameter estimate is:
$$\theta^{(k+1)} = \theta^{(k)} - \eta\nabla\mathcal{L}(\theta^{(k)}) + \alpha\Delta\theta^{(k)}$$
where
$$\Delta\theta^{(k)} = \theta^{(k)} - \theta^{(k-1)}$$
and α is a momentum parameter in the interval $[0, 1]$. Adding momentum to the method of gradient descent can sometimes improve the rate of convergence.

We may opt to use a gradient descent method for this problem in order to be more confident that our algorithm will converge to the global minimum of the negative-log-likelihood function. However, this comes at the cost of computing many more iterations than Newton's method. Naturally, there is an interest in making some modification to the algorithm which exploits the quickness of Newton's method while preserving the certainty of convergence, regardless of the value of the initial guess of the parameter.

2 Discussion

A suggested solution to mitigate the divergence of Newton's method is to *begin* the algorithm by calculating a certain number of iterations of gradient descent, then *switch* to updating the parameter with Newton's method. This is done in the hopes that the gradient descent will bring the estimates sufficiently close to the global minimum such that Newton's method will converge.

There are additional degrees of freedom to consider with the use of gradient descent, such as how large to set the learning rate η and whether or not to add a momentum term. This investigation will explore the use of gradient descent prior to Newton's method for logistic regression, as well as whether or not momentum aids in the convergence/computational cost of the algorithm.

3 Implementation

In order to compare methods for MLE optimization in logistic regression, a synthetic dataset was created using the `make_classification` function from the `datasets` module of `sklearn` in Python. The dataset was generated to have 1000 observations of 20 features used to predict y, a binary response variable.

Iterative methods for computing the maximum-likelihood-estimator of θ^* were also implemented in Python. The gradient of the negative-log-likelihood function was user-defined as `grad(X, theta)`. The method of gradient descent was implemented using a simple for-loop program, repeating the parameter update:

```
for i in range(1, max_iter+1):
    # Compute the gradient of the negative-log-likelihood function at the
    # current value of the parameter estimate.
    gradl = grad(X, y, theta)
    # Update the parameter estimate
    theta = theta - eta*gradl
    # Compute the negative-log-likelihood of the current parameter estimate
    losses.append(log_loss(y,sig(X @ theta)))
```

The implementation of gradient descent with momentum was nearly identical, aside from including the calculation of the momentum term for each iteration:

```
gradl = grad(X, y, theta)
# Update theta, including the addition of momentum
theta_new = theta - eta*gradl + alpha*momentum
# Calculate the new momentum term
momentum = theta_new - theta
theta = theta_new
```

Implementation of Newton's method used a similar for-loop structure, but this time with the addition of a user-defined function `h(X, theta)` for calculating the Hessian of the negative-log-likelihood function:

```
for i in range(1, max_iter+1):
    gradl = grad(X, y, theta)
    # Update theta using Newton's method involving the Hessian matrix
    theta = theta - npl.inv(h(X,theta)) @ gradl
    losses.append(log_loss(y,sig(X@theta)))
```

4 Results

A baseline calculation determined that each element of the optimum parameter θ^* lied in the interval $[-1.5, 1.5]$. However, when an initial guess $\theta^{(0)}$ was generated by sampling each element uniformly from the interval $[-1, 1]$, the parameter estimates from the unmodified Newton's method regularly diverged to infinity. Thus, the use of Newton's method alone was not suitable for estimating θ^*.

Before implementing a combination of Newton's method and gradient descent, it was desired to know if adding momentum improved the rate of convergence to the optimal parameter.

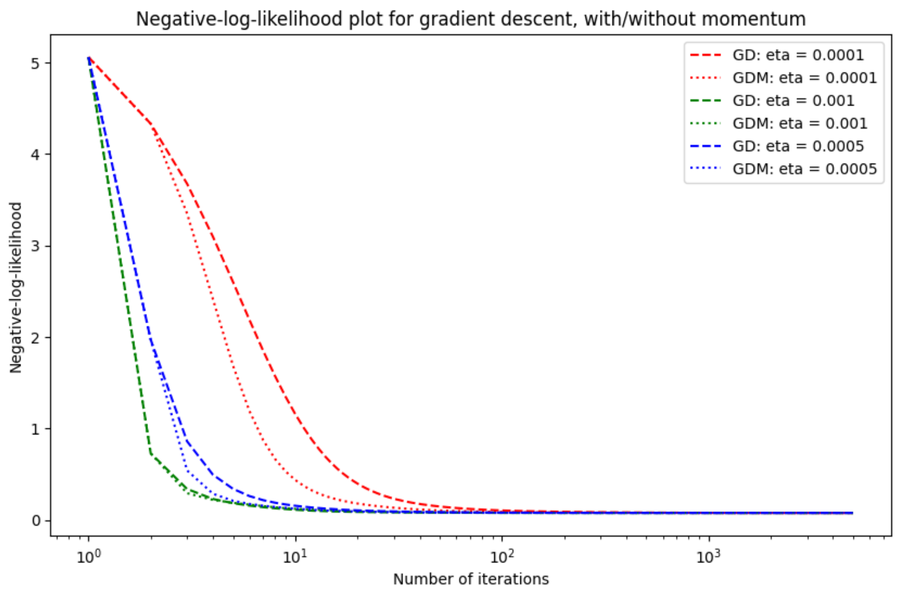

Figure 1: Plots the negative-log-likelihood against the number of iterations at three values of learning rate η. Unaltered gradient descent is shown with a dashed line, while gradient descent with momentum is shown with a dotted line.

These results indicate that for each value of learning rate η, adding momentum tended to help the gradient descent algorithm approach the optimum value with fewer iterations. Since we want to enter a close neighborhood of the global minimizer with the fewest possible iterations of gradient descent, we choose to add momentum for the investigation of switching to Newton's method.

The next results were obtained by calculating 10 iterations of gradient descent with momentum, then switching to Newton's method for the remaining iterations. This is compared with the algorithm using only gradient descent for three different values of the initial guess $\theta^{(0)}$.

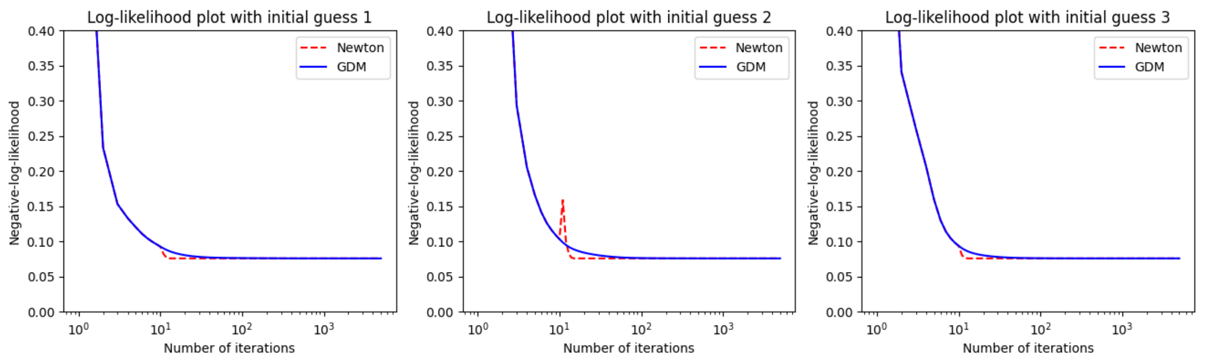

Figure 2: Negative-log-likelihood is plotted against number of iterations for each of three initial guesses generated uniformly from $[0, 1]$.

Another metric of interest is the runtime taken to converge within a certain tolerance. For the following tests, "convergence" was defined as the first iteration at which $\|\nabla\mathcal{L}(\theta^{(k)})\|_2 < 10^{-6}$.

Index of initial guess	**GDM runtime (s)**	**Iters.**	**GDM + Newton runtime (s)**	**Iters.**
1	2.7556	4247	0.04298	27
2	2.6525	4185	0.05407	29
3	2.2282	3838	0.04387	27

Table 1: Displays the runtime and number of iterations required for convergence for each method. A row is included for each choice of initial guess.

5 Conclusion

For each of the three randomly generated initial guesses, initially running just 10 iterations of gradient descent with momentum allowed Newton's method to converge. From Figure 2, it is visible that once the algorithm started iterating with Newton's method, the rate of convergence was very fast and the negative-log-likelihood decreased rapidly. Table 1 also confirms that the method combining GDM with Newton's method took far less time and iterations to converge to a desired tolerance. This investigation has shown me that Newton's method (when it converges) has high potential for speeding up iterative optimization algorithms. The results also show that appropriate variations on gradient descent are crucial if you want efficient algorithms with reasonable runtimes.

6 References

1. Arnold, T., Kane, M., & Lewis, B. W. (2018). Exponential families. In A Computational Approach to Statistical Learning (pp. 128–131). CRC Press.

2. Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Optimization Using Gradient Descent. In Mathematics for Machine Learning (1st ed., pp. 227–233). Cambridge University Press.