Algorithms for maximum-likelihood logistic regression

Thomas P. Minka

September 19, 2003

Abstract

Logistic regression is a workhorse of statistics and is closely related to methods used in Machine Learning including the Perceptron and the Support Vector Machine. This note reviews seven different algorithms for finding the maximum-likelihood estimate. Iterative Scaling is shown to apply under weaker conditions than usually assumed. A modified iterative scaling algorithm is also derived, which is equivalent to the algorithm of Collins et al. (2000). The best performers in terms of running time are the line search algorithms and Newton-type algorithms, which far outstrip Iterative Scaling.

1 Introduction

The logistic regression model is

$$p(y = \pm 1|\mathbf{x}, \mathbf{w}) = \sigma(y\mathbf{w}^{\mathrm{T}}\mathbf{x}) = \frac{1}{1 + \exp(-y\mathbf{w}^{\mathrm{T}}\mathbf{x})} \tag{1}$$

It can be used for binary classification or for predicting the certainty of a binary outcome. See Cox & Snell (1970) for the use of this model in statistics. This note focuses only on computational issues related to maximum-likelihood estimation. Given a data set $(\mathbf{X}, \mathbf{y}) = [(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)]$, we want to find the parameter vector $\mathbf{w}$ which maximizes the log-likelihood:

$$l(\mathbf{w}) = -\sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) \tag{2}$$

The gradient of the log-likelihood is

$$\mathbf{g} = \nabla_{\mathbf{w}} l(\mathbf{w}) = \sum_i (1 - \sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i\mathbf{x}_i \tag{3}$$

Gradient descent using (3) resembles the Perceptron learning algorithm, except that it will always converge for a suitable step size, regardless of whether the classes are separable.

The Hessian of the log-likelihood is

$$\mathbf{H} = \frac{\mathrm{d}^2 l(\mathbf{w})}{\mathrm{d}\mathbf{w}\mathrm{d}\mathbf{w}^{\mathrm{T}}} = -\sum_i \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))\mathbf{x}_i\mathbf{x}_i^{\mathrm{T}} \tag{4}$$

which in matrix form can be written

$$a_{ii} = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) \quad (5)$$

$$\mathbf{H} = -\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} \quad (6)$$

Note that the Hessian does not depend on how the $\mathbf{x}$'s are labeled. It is nonpositive definite, which means $l(\mathbf{w})$ is convex.

Suppose the prior for $\mathbf{w}$ is uniform. Then when the classes are nonseparable, the posterior is approximately Gaussian:

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \approx N(\mathbf{w};\hat{\mathbf{w}},-\mathbf{H}^{-1}) \quad (7)$$

$$\hat{\mathbf{w}} = \operatorname{argmax} \prod_i p(y_i|\mathbf{x}_i,\mathbf{w}) \quad (8)$$

So the model likelihood can be approximated by

$$p(\mathbf{y}|\mathbf{X}) \approx p(\mathbf{y},\hat{\mathbf{w}}|\mathbf{X})(2\pi)^{d/2}\,|-\mathbf{H}|^{-1/2} \quad (9)$$

where d is the dimensionality of $\mathbf{w}$.

Unfortunately, when the classes are separable, then the posterior is improper: the magnitude of $\mathbf{w}$ can be increased without bound. This suggests a nonuniform prior on $\mathbf{w}$, such as:

$$p(\mathbf{w}) \sim \mathcal{N}(\mathbf{0}, v\mathbf{I}) \quad (10)$$

Performing MAP with this prior instead of ML gives a "regularized" estimate (Nigam et al., 1999).

2 Newton's method

If we define

$$z_i = \mathbf{x}_i^{\mathrm{T}}\mathbf{w}_{\mathrm{old}} + \frac{(1-\sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i}{a_{ii}} \quad (11)$$

Then a Newton step is

$$\mathbf{w}_{\mathrm{new}} = \mathbf{w}_{\mathrm{old}} + (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}})^{-1}\sum_i (1-\sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i\mathbf{x}_i \quad (12)$$

$$= (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}})^{-1}\mathbf{X}\mathbf{A}(\mathbf{X}^{\mathrm{T}}\mathbf{w}_{\mathrm{old}} + \left[\frac{(1-\sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i}{a_{ii}}\right]) \quad (13)$$

$$= (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}})^{-1}\mathbf{X}\mathbf{A}\mathbf{z} \quad (14)$$

which we recognize as the solution to a weighted least squares problem. This efficient implementation of Newton's method is called Iteratively Reweighted Least Squares. It requires $n > d$ and takes $O(nd^2)$ time per iteration.

3 Line search methods

Because the likelihood is convex, we can also optimize each w_k alternately. A coordinate-wise Newton update is

$$w_k^{new} = w_k^{old} + \frac{-v^{-1} w_k^{old} + \sum_i (1 - \sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)) y_i x_{ik}}{v^{-1} + \sum_i a_{ii} x_{ik}^2} \tag{15}$$

(Use $v = \infty$ for maximum-likelihood.) To implement this efficiently, note that $\mathbf{w}^{\mathrm{T}} \mathbf{x}_i$ for all i can be incrementally updated via

$$(\mathbf{w}^{new})^{\mathrm{T}} \mathbf{x}_i = (\mathbf{w}^{old})^{\mathrm{T}} \mathbf{x}_i + (w_k^{new} - w_k^{old}) x_{ik} \tag{16}$$

Using this trick, it takes $O(n)$ time to update w_k and thus $O(nd)$ time to update all components of $\mathbf{w}$.

This idea can be generalized to updating in an arbitrary direction $\mathbf{u}$. Assuming $\|\mathbf{u}\| = 1$, the Newton step along $\mathbf{u}$ is

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \frac{\mathbf{g}^{\mathrm{T}} \mathbf{u}}{v^{-1} - \mathbf{u}^{\mathrm{T}} \mathbf{H} \mathbf{u}} \mathbf{u} \tag{17}$$

To implement this efficiently, use the expansion

$$-\mathbf{u}^{\mathrm{T}} \mathbf{H} \mathbf{u} = \sum_i a_{ii} (\mathbf{u}^{\mathrm{T}} \mathbf{x}_i)^2 \tag{18}$$

Each update takes $O(nd)$ time, which is more expensive than the $O(n)$ time required for updating along a coordinate direction. However, this method can be faster if we choose good update directions. One approach is to use the gradient as the update direction. Even better is to use the so-called *conjugate gradient* rule, which subtracts the previous update direction from the gradient:

$$\mathbf{u} = \mathbf{g} - \mathbf{u}^{old} \beta \tag{19}$$

There are various heuristic ways to set the scale factor β (Bishop, 1995). The method which seems to work best in practice is the Hestenes-Stiefel formula:

$$\beta = \frac{\mathbf{g}^{\mathrm{T}} (\mathbf{g} - \mathbf{g}^{old})}{(\mathbf{u}^{old})^{\mathrm{T}} (\mathbf{g} - \mathbf{g}^{old})} \tag{20}$$

4 Dual line search

Jaakkola & Haussler (1999) have shown that the MAP objective has a simple dual formulation in terms of Legendre transformations. The dual representation of the sigmoid is

$$\log \sigma(x) = \min_{\lambda} \lambda x - H(\lambda) \tag{21}$$

$$H(\lambda) = -\lambda \log \lambda - (1 - \lambda) \log(1 - \lambda) \tag{22}$$

With a Gaussian prior on $\mathbf{w}$, the objective function is

$$
\begin{aligned}
p(\mathbf{w}) &\sim \mathcal{N}(\mathbf{0}, v\mathbf{I}) && (23)\\
J(\mathbf{w}) &= \sum_i \log \sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i) - \frac{1}{2v}\mathbf{w}^{\mathrm{T}}\mathbf{w} && (24)\\
&= \min_{\boldsymbol{\lambda}} J(\mathbf{w}, \boldsymbol{\lambda}) && (25)\\
J(\mathbf{w}, \boldsymbol{\lambda}) &= \sum_i \lambda_i y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i - H(\lambda_i) - \frac{1}{2v}\mathbf{w}^{\mathrm{T}}\mathbf{w} && (26)
\end{aligned}
$$

We can now fold in the maximum over $\mathbf{w}$ to get

$$
\begin{aligned}
\mathbf{w}(\boldsymbol{\lambda}) &= v \sum_i \lambda_i y_i \mathbf{x}_i && (27)\\
J(\boldsymbol{\lambda}) &= \max_{\mathbf{w}} J(\mathbf{w}, \boldsymbol{\lambda}) && (28)\\
&= \frac{v}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i - \sum_i H(\lambda_i) && (29)
\end{aligned}
$$

This objective only involves inner products between data points. The derivatives are

$$
\begin{aligned}
\frac{dJ(\boldsymbol{\lambda})}{d\lambda_i} &= v y_i \sum_j \lambda_j y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i - \log \frac{1-\lambda_i}{\lambda_i} && (30)\\
&= y_i \hat{\mathbf{w}}^{\mathrm{T}} \mathbf{x}_i - \log \frac{1-\lambda_i}{\lambda_i} && (31)\\
\frac{d^2 J(\boldsymbol{\lambda})}{d\lambda_i^2} &= v \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i + \frac{1}{\lambda_i(1-\lambda_i)} && (32)\\
\frac{d^2 J(\boldsymbol{\lambda})}{d\lambda_i d\lambda_j} &= v y_i y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i && (33)
\end{aligned}
$$

Newton's method on the full $\boldsymbol{\lambda}$ vector would require inverting the $n \times n$ Hessian matrix

$$
\mathbf{H} = v\,\mathrm{diag}(\mathbf{y})\mathbf{X}^{\mathrm{T}}\mathbf{X}\,\mathrm{diag}(\mathbf{y}) + \mathrm{diag}\left(\frac{1}{\lambda_i(1-\lambda_i)}\right) \qquad (34)
$$

Jaakkola & Haussler (1999) recommend instead to update one λ_i at a time. The coordinate-wise Newton update is

$$
\lambda_i^{new} = \lambda_i - \frac{y_i \hat{\mathbf{w}}^{\mathrm{T}} \mathbf{x}_i - \log \frac{1-\lambda_i}{\lambda_i}}{v \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i + \frac{1}{\lambda_i(1-\lambda_i)}} \qquad (35)
$$

If this update would take λ_i outside the region $[0, 1]$, then we stop it at the endpoint. Computing the full set of inner products $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ takes $O(n^2 d)$ time, and each iteration (updating all λ_i's) takes $O(n^2)$ time.

This algorithm technically cannot compute an MLE since it requires a proper prior. A workaround is to make v very large. However, the convergence rate depends strongly on v—many iterations are required if v is large. So the best scheme is to start with small v and gradually "anneal" the prior to uniform.

5 Böhning's method

Böhning's method is a quasi-Newton method, i.e. a Newton method using a different matrix $\tilde{\mathbf{H}}$ in place of the Hessian matrix $\mathbf{H}$. Böhning (1999) has shown that the convergence of quasi-Newton is guaranteed as long as $\tilde{\mathbf{H}} \leq \mathbf{H}$ in the sense that $\mathbf{H} - \tilde{\mathbf{H}}$ is positive definite. He suggests the matrix

$$\tilde{\mathbf{H}} = -\frac{1}{4}\mathbf{X}\mathbf{X}^{\mathrm{T}} \tag{36}$$

This matrix must be less than $\mathbf{H}$ because $\frac{1}{4} \geq \sigma(x)(1-\sigma(x))$ for any x and therefore $\frac{1}{4}\mathbf{I} \geq \mathbf{A}$. Because $\tilde{\mathbf{H}}$ does not depend on $\mathbf{w}$, we can precompute its inverse—or even simpler its LU decomposition. The resulting algorithm is:

Setup Compute the LU decomposition of $\tilde{\mathbf{H}}$.

Iterate $\mathbf{w}^{new} = \mathbf{w}^{old} - \tilde{\mathbf{H}}^{-1}\mathbf{g}$
where $\tilde{\mathbf{H}}^{-1}\mathbf{g}$ is computed by back-substitution.

This algorithm has $O(nd^2)$ setup cost and $O(nd + d^2)$ cost per iteration ($O(nd)$ for computing the gradient and $O(d^2)$ for back-substitution).

A slightly better algorithm is possible by using

$$\tilde{\mathbf{H}} = -b(\mathbf{w})\mathbf{X}\mathbf{X}^{\mathrm{T}} \tag{37}$$

$$b(\mathbf{w}) = \max_i \sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)(1 - \sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)) \tag{38}$$

which also satisfies $\tilde{\mathbf{H}} \leq \mathbf{H}$. But the difference is minimal in practice.

6 Iterative Scaling

Iterative scaling is a lower bound method for finding the likelihood maximum. It produces a lower bound which is additive in the parameters w_k, which means we have the option to update one or all of them at each step. The algorithm requires that all feature values are positive: $x_{ik} > 0$. Define $s = \max_i \sum_k x_{ik}$. Unlike most derivations of iterative scaling, we will not require $\sum_k x_{ik} = 1$.

Iterative scaling is based on the following two bounds:

$$-\log(x) \geq 1 - \frac{x}{x_0} - \log(x_0) \qquad \text{for any } x_0 \tag{39}$$

$$-\exp(-\sum_k q_k w_k) \geq -\sum_k q_k \exp(-w_k) - (1 - \sum_k q_k) \tag{40}$$

$$\text{for any } q_k > 0 \text{ satisfying } \sum_k q_k \leq 1$$

The second bound comes from Jensen's inequality applied to the function e^{-x}:

$$\exp(-\sum_k q_k w_k) \leq \sum_k q_k \exp(-w_k) \tag{41}$$

$$\text{if } \sum_k q_k = 1 \tag{42}$$

Now let some of the $w_k = 0$ to get

$$\exp(-\sum_k q_k w_k) \leq \sum_k q_k \exp(-w_k) + (1 - \sum_k q_k) \tag{43}$$

$$\text{if } \sum_k q_k \leq 1 \tag{44}$$

Start by writing the likelihood in an asymmetric way:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i|y_i=1} \frac{\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \prod_{i|y_i=-1} \frac{1}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \tag{45}$$

Applying the first bound at the current parameter values $\mathbf{w}_0$, we obtain that the log-likelihood function is bounded by

$$\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \sum_{i|y_i=1} \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - \sum_i \log(1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) \tag{46}$$

$$\geq \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_{i|y_i=1} (\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i + \sum_i (1 - \frac{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1 + \exp(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i)}) \tag{47}$$

$$= \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_{i|y_i=1} (\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i + \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}} x_i)(1 - \exp((\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i)) \tag{48}$$

Maximizing this bound over $\mathbf{w}$ is still too hard. So we apply the second bound, with $q_k = x_{ik}/s$, remembering that $x_{ik} > 0$:

$$
\begin{aligned}
-\exp((\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i) &= -\exp(\sum_k (w_k - w_{0k})x_{ik}) && (49)\\
&\geq -\sum_k \frac{x_{ik}}{s}\exp((w_k - w_{0k})s) - (1 - \sum_k \frac{x_{ik}}{s}) && (50)
\end{aligned}
$$

Note that s was chosen to ensure $\sum_k q_k \leq 1$. Thanks to this bound, the algorithm reduces to a one-dimensional maximization for each w_k of

$$
g(w_k) = \sum_{i|y_i=1}(w_k - w_{0k})x_{ik} - \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)\sum_k \frac{x_{ik}}{s}\exp((w_k - w_{0k})s) \tag{51}
$$

Zero the gradient with respect to w_k:

$$
\begin{aligned}
\frac{dg(w_k)}{dw_k} &= \sum_{i|y_i=1} x_{ik} - \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}\exp((w_k - w_{0k})s) = 0 && (52)\\
\exp((w_k - w_{0k})s) &= \frac{\sum_{i|y_i=1} x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}} && (53)\\
w_k &= w_{0k} + \frac{1}{s}\log\frac{\sum_{i|y_i=1} x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}} && (54)
\end{aligned}
$$

This is one possible iterative scaling update. Note that we could have written a different asymmetric likelihood:

$$
p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \prod_{i|y_i=1}\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}\prod_{i|y_i=-1}\frac{\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \tag{55}
$$

which would have led to the update

$$
\exp((w_k - w_{0k})s) = \frac{\sum_i (1-\sigma(\mathbf{w}_0^{\mathrm{T}}x_i))x_{ik}}{\sum_{i|y_i=-1} x_{ik}} \tag{56}
$$

So the fair thing to do is combine the two updates:

$$
\exp((w_k - w_{0k})s) = \frac{\sum_{i|y_i=1} x_{ik}}{\sum_{i|y_i=-1} x_{ik}}\frac{\sum_i (1-\sigma(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i))x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i)x_{ik}} \tag{57}
$$

This is the iterative scaling update used in practice (Nigam et al., 1999; Collins et al., 2000), and it converges faster than either of the two asymmetric updates. Note that the first term is constant throughout the iteration and can be precomputed. The running time is $O(nd)$ per iteration.

7 Modified Iterative Scaling

We can get a different iterative scaling algorithm by applying the same bounds to the symmetric likelihood:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_i \frac{1}{1 + \exp(-y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)} \tag{58}$$

Applying the first bound at the current parameter values $\mathbf{w}_0$, we obtain that the log-likelihood function is bounded by

$$\begin{aligned}
\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= -\sum_i \log(1 + \exp(-y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)) && (59)\\
&\geq \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_i (1 - \frac{1 + \exp(-y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i)}{1 + \exp(-y_i \mathbf{w}_0^{\mathrm{T}} \mathbf{x}_i)}) && (60)\\
&= \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i))(1 - \exp(-y_i (\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}} \mathbf{x}_i)) && (61)
\end{aligned}$$

Maximizing this bound over $\mathbf{w}$ is still too hard. So we apply the second bound with $q_k = x_{ik}/s$, remembering that $x_{ik} > 0$:

$$-\exp(-y_i (\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}} \mathbf{x}_i) \geq -\sum_k \frac{x_{ik}}{s} \exp(-y_i s(w_k - w_{0k})) - (1 - \sum_k \frac{x_{ik}}{s}) \tag{62}$$

The algorithm reduces to a one-dimensional maximization for each w_k of

$$g(w_k) = -\sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) \sum_k \frac{x_{ik}}{s} \exp(-y_i s(w_k - w_{0k})) \tag{63}$$

The gradient with respect to w_k is

$$\frac{dg(w_k)}{dw_k} = \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) y_i x_{ik} \exp(-y_i s(w_k - w_{0k})) = 0 \tag{64}$$

Multiply both sides by $\exp(-s(w_k - w_{0k}))$ and solve for w_k to get

$$\exp(2s(w_k - w_{0k})) = \frac{\sum_{i|y_i=1} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) x_{ik}}{\sum_{i|y_i=-1} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) x_{ik}} \tag{65}$$

To allow negative feature values x_{ik}, define $s = \max_i \sum_k |x_{ik}|$ and use $q_k = |x_{ik}|/s$ to get

$$\begin{aligned}
g(w_k) &= -\sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) \sum_k \frac{|x_{ik}|}{s} \exp(-y_i \mathrm{sign}(x_{ik}) s(w_k - w_{0k})) && (66)\\
\frac{dg(w_k)}{dw_k} &= \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) y_i x_{ik} \exp(-y_i \mathrm{sign}(x_{ik}) s(w_k - w_{0k})) = 0 && (67)\\
\exp(2s(w_k - w_{0k})) &= \frac{\sum_{i|y_i x_{ik} > 0} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) |x_{ik}|}{\sum_{i|y_i x_{ik} < 0} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) |x_{ik}|} && (68)
\end{aligned}$$

This update rule was also given by Collins et al. (2000), using a boosting argument. This algorithm will be called Modified Iterative Scaling. The cost is $O(nd)$ per iteration.

8 Results

The missing factor in the above analysis is the number of iterations needed by each algorithm. This section compares the algorithms empirically on real and simulated data. All algorithms are started at $\mathbf{w} = \mathbf{0}$ and performance is measured according to the log-likelihood value achieved. The results are not substantially affected by using a random starting point instead of $\mathbf{w} = \mathbf{0}$. Cost is measured by total floating-point operations (FLOPS) using the `flops` command in Matlab. This is more meaningful than comparing the number of iterations or wall-clock time. Each algorithm was implemented to minimize its FLOP count, e.g. by precomputing common subexpressions.

The first experiment repeats the setup of Collins et al. (2000). For a given dimensionality d, feature vectors are drawn from a standard normal: $\mathbf{x} \sim \mathcal{N}(0, \mathbf{I}_d)$. A true parameter vector is chosen randomly on the surface of the d-dimensional sphere with radius $\sqrt{2}$. Finally, the feature vectors are classified randomly according to the logistic model. Due to the choice of $\mathbf{w}$, about 16% of the data will be mislabeled. Each of the algorithms is then run to find the MLE for $\mathbf{w}$ (which is not necessarily the true $\mathbf{w}$). In this experiment, Iterative Scaling cannot be run since some feature values are negative (but see the next experiment). The dual line search method was run with prior variance $v = 10$ to get an approximate MLE.

Figure 1 shows the result for a typical dataset with $(d = 100, n = 300)$. It really matters which algorithm you use: the difference in cost between the best (CG) and worst (MIS) algorithms is more than two orders of magnitude! The relative performance of the algorithms remains the same for smaller d, and varies little across repeated draws of the dataset. What about bigger problems? Figure 2 shows the result for a typical dataset with $(d = 500, n = 1500)$. The differences simply get bigger. The cost difference between CG and MIS is now more than three orders of magnitude.

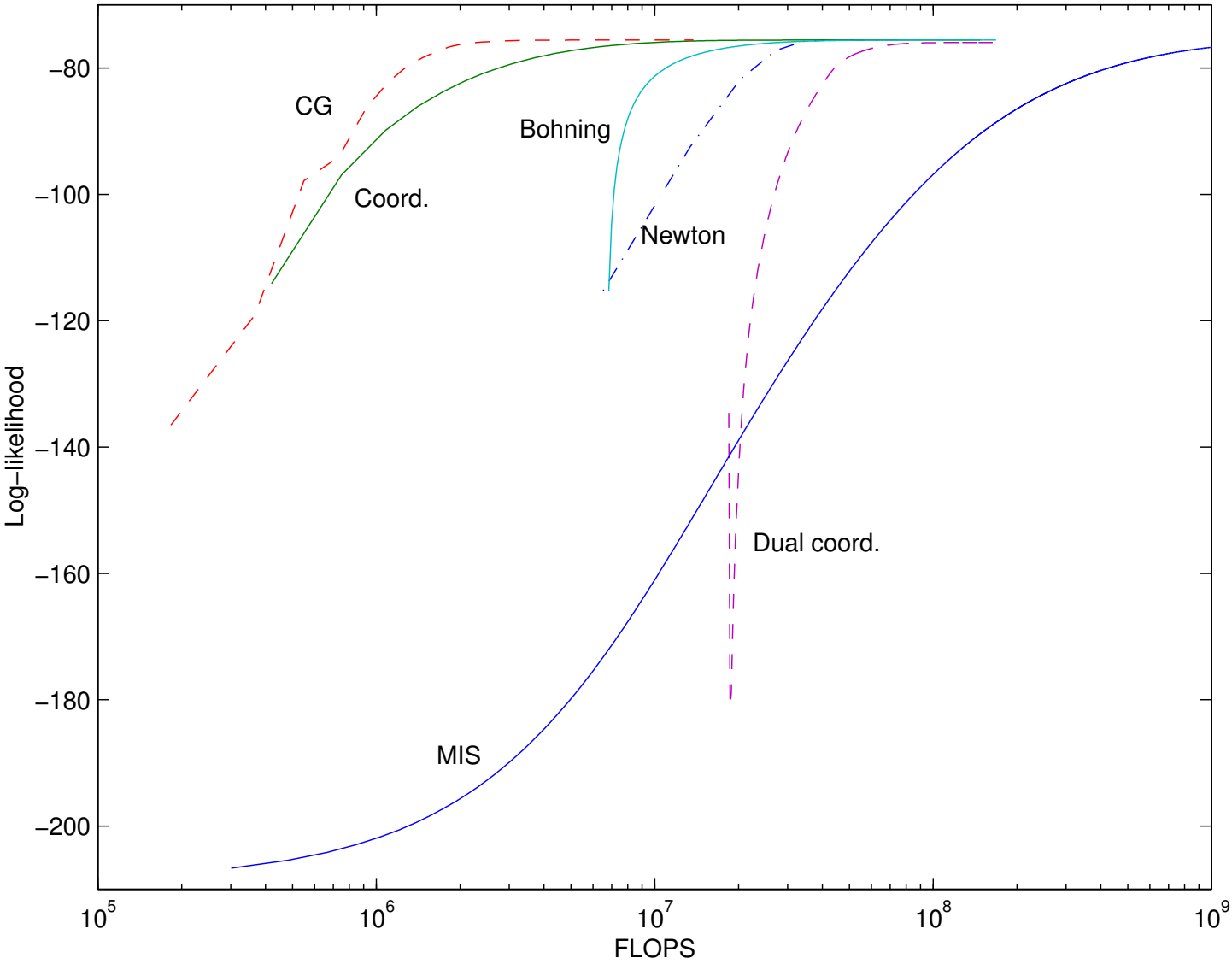

Figure 1: Cost vs. performance of six logistic regression algorithms. The dataset had 300 points in 100 dimensions. "CG" is conjugate gradient (section 3), "Coord." is coordinate-wise Newton, and "MIS" is modified iterative scaling. CG also has the smallest clock time in Matlab.

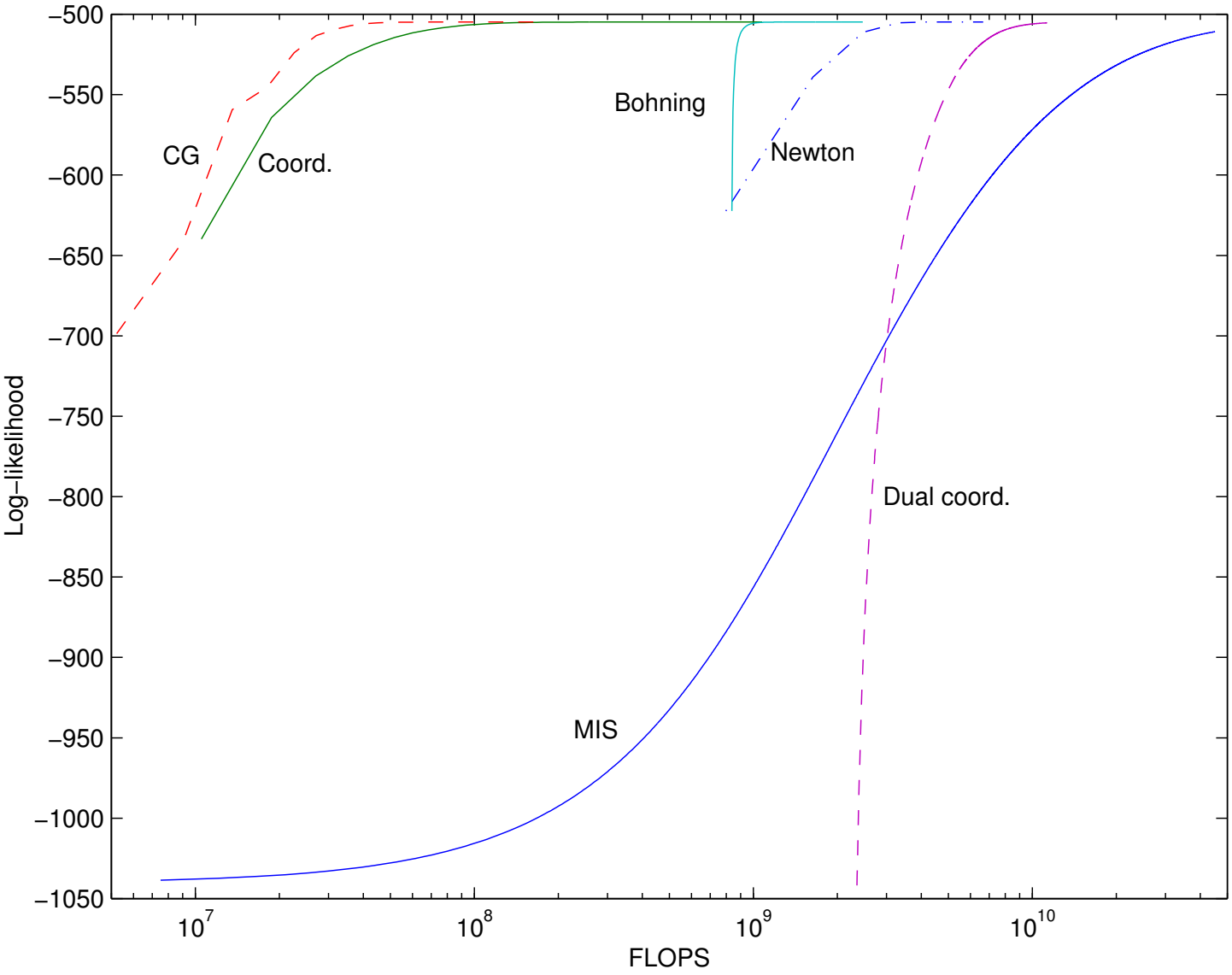

Figure 2: Cost vs. performance of six logistic regression algorithms. The dataset had 1500 points in 500 dimensions. CG took 5 seconds, Newton took 2 minutes, and MIS took 1.8 hours to reach the last point plotted.

The previous experiment had independently distributed features. The next experiment uses highly correlated features. A dataset is first generated according to the previous experiment and then modified by adding 10 to all feature values x_{ik}. This introduces significant correlation, in the sense that $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ has significant off-diagonal elements. To make the labels consistent with this shift, an extra feature x_{i0} is added with value 1 and classification weight $w_0 = -10 \sum_{i=1}^{d} w_i$. This ensures that $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i$ is unchanged by the shift. Under this setup, we expect the coordinate-wise algorithms to perform poorly. Thanks to the shift, the features are all positive, so we can run Iterative Scaling this time.

Figure 3 shows the result. It suggests two things: (1) it is hard to beat Newton's method for correlated data, and (2) you should decorrelate your data before running logistic regression. For this data it is easy to decorrelate by subtracting the mean of each feature, but in other cases the correlation may be more subtle. The dual line search method performs particularly poorly for correlated data. Its likelihood curve is not monotonic because it is optimizing a dual function, not the original likelihood. The best result for it was obtained with $v = 1$ (shown).

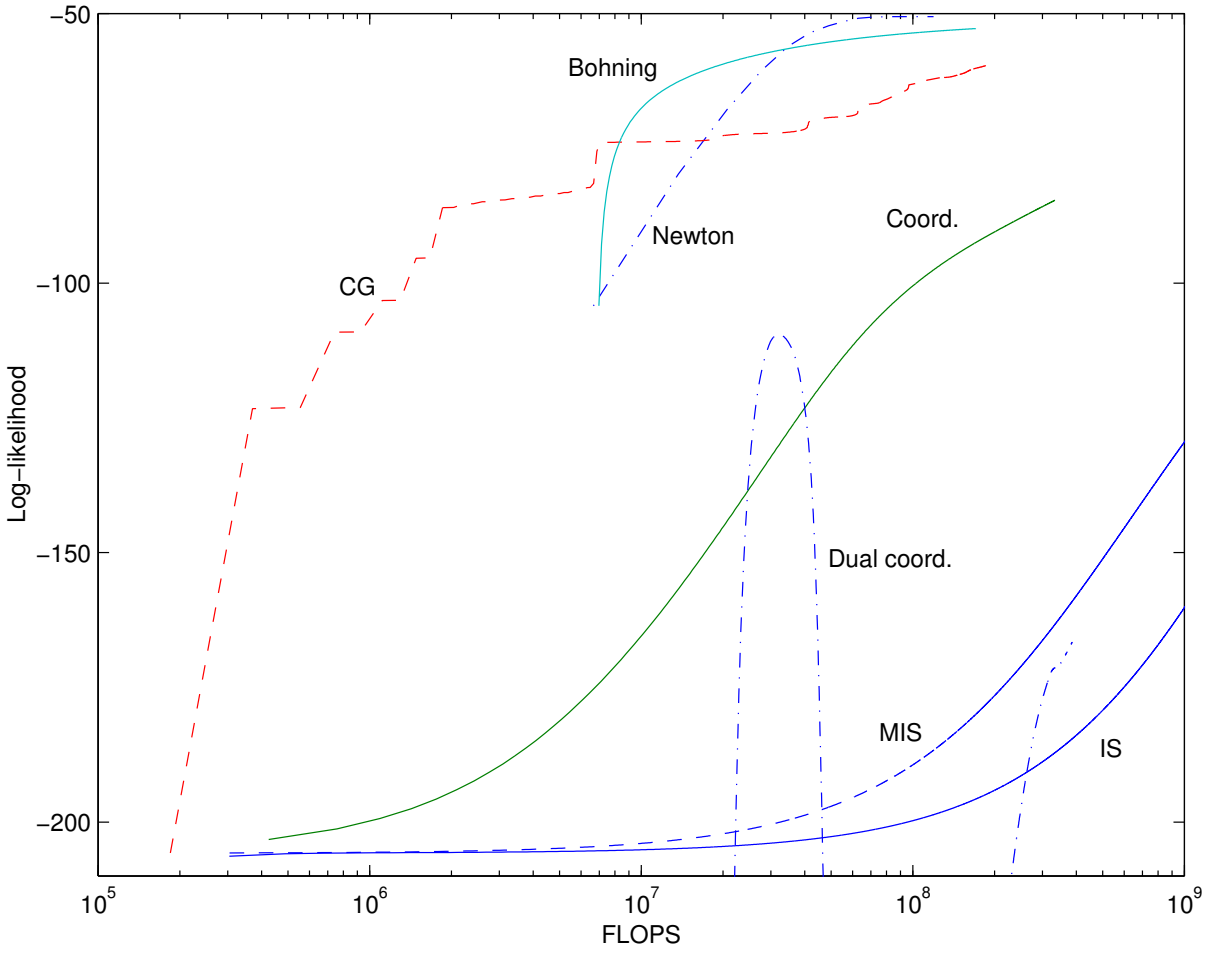

Figure 3: Cost vs. performance of logistic regression algorithms on correlated data. The dataset had 300 points in 100 dimensions.

The third experiment simulates a document classification problem. It is designed to be as favorable as possible to Iterative Scaling. A multinomial classifier has

$$p(y=1|\mathbf{x},\mathbf{p},\mathbf{q}) = \frac{\prod_k p_k^{x_{ik}}}{\prod_k p_k^{x_{ik}} + \prod_k q_k^{x_{ik}}} = \sigma(\sum_k x_{ik} \log \frac{p_k}{q_k}) \quad (69)$$

$$\sigma(x) = \frac{1}{1+\exp(-x)} \quad (70)$$

which can be translated into a logistic regression problem. For the experiment, feature vectors are drawn from a uniform Dirichlet distribution: $\mathbf{x}_i \sim \mathcal{D}(1,...,1)$, which means $x_{ik} > 0$ and $\sum_k x_{ik} = 1$. A true parameter vector is drawn according to $\log \frac{p_k}{q_k}$, where $\mathbf{p}$ and $\mathbf{q}$ have a uniform Dirichlet distribution. Finally, the feature vectors are classified randomly according to the logistic model.

Figure 4 shows the result for a typical dataset with $(d = 100, n = 300)$. Modified Iterative Scaling is better than regular Iterative Scaling, but they are still the worst of the bunch, even when $\sum_k x_{ik} = 1$. Dual line search did not perform well with any fixed value of v so instead v was annealed by adding 100 at each step, starting from $v = 1000$.

References

Bishop, C. (1995). *Neural networks for pattern recognition.* Oxford: Clarendon Press.

Böhning, D. (1999). The lower bound method in probit regression. *Computational Statistics and Data Analysis*, *30*, 13–17.

Collins, M., Schapire, R. E., & Singer, Y. (2000). Logistic regression, AdaBoost and Bregman distances. *COLT.* `http://www.research.att.com/~schapire/papers/breg-dist.ps.gz`.

Cox, D. R., & Snell, E. J. (1970). *The analysis of binary data.* Chapman and Hall.

Jaakkola, T., & Haussler, D. (1999). Probabilistic kernel regression models. *Seventh International Workshop on Artificial Intelligence and Statistics.* `http://www.ai.mit.edu/~tommi/papers.html`.

Nigam, K., Lafferty, J., & McCallum, A. (1999). Using Maximum Entropy for text classification. *IJCAI'99 Workshop on Information Filtering.* `http://www.cs.cmu.edu/~mccallum/`.

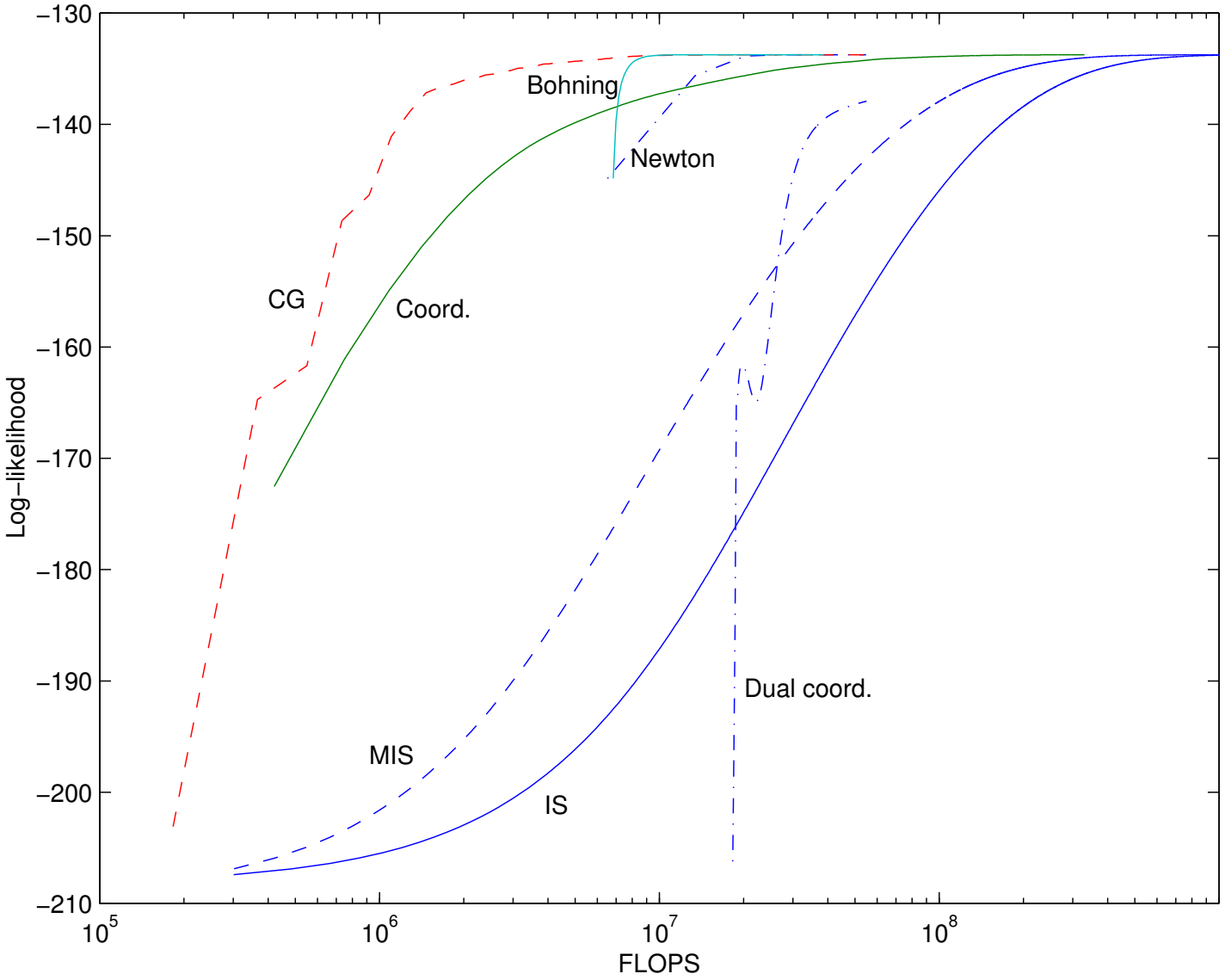

Figure 4: Cost vs. performance of logistic regression algorithms on positive data. The dataset had 300 points in 100 dimensions.