A comparison of numerical optimizers for logistic regression

Thomas P. Minka

October 22, 2003 (revised Mar 26, 2007)

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 compares eight different algorithms for computing the maximum a-posteriori parameter estimate. A full derivation of each algorithm is given. In particular, a new derivation of Iterative Scaling is given which applies more generally than the conventional one. A new derivation is also given for the Modified Iterative Scaling algorithm of Collins et al. (2002). Most of the algorithms operate in the primal space, but can also work in dual space. All algorithms are compared in terms of computational complexity by experiments on large data sets. The fastest algorithms turn out to be conjugate gradient ascent and quasi-Newton algorithms, which far outstrip Iterative Scaling and its variants.

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 or more generally maximum a-posteriori (MAP) estimation. A common prior to use with MAP is:

$$p(\mathbf{w}) \;\sim\; \mathcal{N}(\mathbf{0}, \lambda^{-1}\mathbf{I}) \tag{2}$$

Using $\lambda > 0$ gives a "regularized" estimate of $\mathbf{w}$ which often has superior generalization performance, especially when the dimensionality is high (Nigam et al., 1999).

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:

$$l(\mathbf{w}) = -\sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) - \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{3}$$

The gradient of this objective is

$$\mathbf{g} = \nabla_{\mathbf{w}} l(\mathbf{w}) = \sum_i (1 - \sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i\mathbf{x}_i - \lambda\mathbf{w} \tag{4}$$

Gradient descent using (4) 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 objective is

$$\mathbf{H} = \frac{\mathrm{d}^2 l(\mathbf{w})}{\mathrm{d}\mathbf{w}\mathrm{d}\mathbf{w}^{\mathrm{T}}} = -\sum_i \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))\mathbf{x}_i\mathbf{x}_i^{\mathrm{T}} - \lambda\mathbf{I} \tag{5}$$

which in matrix form can be written

$$\begin{aligned} a_{ii} &= \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) & (6)\\ \mathbf{H} &= -\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} - \lambda\mathbf{I} & (7)\end{aligned}$$

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.

For sufficiently large λ, the posterior for $\mathbf{w}$ will be approximately Gaussian:

$$\begin{aligned} p(\mathbf{w}|\mathbf{X},\mathbf{y}) &\approx N(\mathbf{w};\hat{\mathbf{w}},-\mathbf{H}^{-1}) & (8)\\ \hat{\mathbf{w}} &= \text{argmax}p(\mathbf{w})\prod_i p(y_i|\mathbf{x}_i,\mathbf{w}) & (9)\end{aligned}$$

So the model likelihood can be approximated by

$$p(\mathbf{y}|\mathbf{X}) \approx p(\mathbf{y}|\mathbf{X},\hat{\mathbf{w}})p(\hat{\mathbf{w}})(2\pi)^{d/2}\,|-\mathbf{H}|^{-1/2} \tag{10}$$

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

2 Newton's method

If we define

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

Then a Newton step is

$$\begin{aligned} \mathbf{w}_{\text{new}} &= \mathbf{w}_{\text{old}} + (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\left(\sum_i (1-\sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i\mathbf{x}_i - \lambda\mathbf{w}\right) & (12)\\ &= (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{A}(\mathbf{X}^{\mathrm{T}}\mathbf{w}_{\text{old}} + \left[\frac{(1-\sigma(y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))y_i}{a_{ii}}\right]) & (13)\\ &= (\mathbf{X}\mathbf{A}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{A}\mathbf{z} & (14)\end{aligned}$$

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

3 Coordinate ascent

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

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

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 $\mathbf{w}$.

4 Conjugate gradient ascent

The idea of the last section can be generalized to updating in an arbitrary direction $\mathbf{u}$. The Newton step along $\mathbf{u}$ is

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

To implement this efficiently, use the expansion

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

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:

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

This formula is derived in the following way. When the surface is quadratic with Hessian $\mathbf{H}$, we want the new search direction to be orthogonal with respect to $\mathbf{H}$: $\mathbf{u}^{\mathrm{T}} \mathbf{H} \mathbf{u}^{old} = 0$, which implies

$$\beta = \frac{\mathbf{g}^{\mathrm{T}} \mathbf{H} \mathbf{u}^{old}}{(\mathbf{u}^{old})^{\mathrm{T}} \mathbf{H} \mathbf{u}^{old}} \tag{21}$$

On a quadratic surface, the pairs $(\mathbf{w}^{old}, \mathbf{g}^{old})$ and $(\mathbf{w}, \mathbf{g})$ must satisfy

$$\mathbf{g} - \mathbf{g}^{old} = \mathbf{H}(\mathbf{w} - \mathbf{w}^{old}) \tag{22}$$

But since $\mathbf{w} = \mathbf{w}^{old} + \alpha\mathbf{u}^{old}$, we obtain

$$\mathbf{g} - \mathbf{g}^{old} = \alpha\mathbf{H}\mathbf{u}^{old} \tag{23}$$

Substituting (23) into (21) gives (20).

5 Fixed-Hessian Newton method

Another way to speed up Newton's method is to approximate the varying Hessian with a fixed matrix $\tilde{\mathbf{H}}$ that only needs to be inverted once. Böhning (1999) has shown that the convergence of this approach is guaranteed as long as $\tilde{\mathbf{H}} \leq \mathbf{H}$ in the sense that $\mathbf{H} - \tilde{\mathbf{H}}$ is positive definite. For maximum-likelihood estimation, he suggests the matrix $\tilde{\mathbf{H}} = -\frac{1}{4}\mathbf{X}\mathbf{X}^{\mathrm{T}}$, which for MAP generalizes to

$$\tilde{\mathbf{H}} = -\frac{1}{4}\mathbf{X}\mathbf{X}^{\mathrm{T}} - \lambda\mathbf{I} \tag{24}$$

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 Cholesky 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 faster-converging algorithm can obtained by ignoring the bound requirement and using line searches. That is, we take $\mathbf{u} = -\tilde{\mathbf{H}}^{-1}\mathbf{g}$ as a search direction and apply (17). In this way, the approximate Hessian is scaled automatically—not by a predefined factor of $\frac{1}{4}$. Because it performs significantly better, this is the version used in experiments.

6 Quasi-Newton

Instead of holding the approximate Hessian fixed, as in Böhning's method, we can allow it to vary *in the inverse domain*. This is the idea behind the more general quasi-Newton algorithms, DFP and BFGS. The algorithm starts with $\tilde{\mathbf{H}}^{-1} = \mathbf{I}$. At each step we update $\mathbf{w}^{new} = \mathbf{w} + \Delta\mathbf{w}$, giving a change in the gradient, $\Delta\mathbf{g} = \mathbf{g}^{new} - \mathbf{g}$. If the function is quadratic, we must have

$$\Delta\mathbf{w} = -\mathbf{H}^{-1}\Delta\mathbf{g} \tag{25}$$

The approximate inverse Hessian, $\tilde{\mathbf{H}}^{-1}$, is repeatedly updated to satisfy this constraint.

In the BFGS algorithm, the update is (Bishop, 1995)

$$b = 1 + \frac{\Delta\mathbf{g}^{\mathrm{T}}\tilde{\mathbf{H}}^{-1}\Delta\mathbf{g}}{\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{g}} \tag{26}$$

$$\tilde{\mathbf{H}}^{-1}_{new} = \tilde{\mathbf{H}}^{-1} + \frac{1}{\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{g}}\left(b\Delta\mathbf{w}\Delta\mathbf{w}^{\mathrm{T}} - \Delta\mathbf{w}\Delta\mathbf{g}^{\mathrm{T}}\tilde{\mathbf{H}}^{-1} - \tilde{\mathbf{H}}^{-1}\Delta\mathbf{g}\Delta\mathbf{w}^{\mathrm{T}}\right) \tag{27}$$

You can verify that $\tilde{\mathbf{H}}^{-1}_{new}$ satisfies (25). To update $\mathbf{w}$, treat $\mathbf{u} = -\tilde{\mathbf{H}}^{-1}\mathbf{g}$ as a search direction and apply (17). This works better than simply setting $\Delta\mathbf{w} = \mathbf{u}$. The cost is $O(d^2 + nd)$ per iteration.

A variant of this algorithm is the so-called "limited-memory" BFGS (Bishop, 1995), where $\tilde{\mathbf{H}}^{-1}$ is not stored but assumed to be $\mathbf{I}$ before each update. This gives the search direction

$$b = 1 + \frac{\Delta\mathbf{g}^{\mathrm{T}}\Delta\mathbf{g}}{\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{g}} \tag{28}$$

$$a_g = \frac{\Delta\mathbf{w}^{\mathrm{T}}\mathbf{g}}{\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{g}} \tag{29}$$

$$a_w = \frac{\Delta\mathbf{g}^{\mathrm{T}}\mathbf{g}}{\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{g}} - ba_g \tag{30}$$

$$\mathbf{u} = -\mathbf{g} + a_w\Delta\mathbf{w} + a_g\Delta\mathbf{g} \tag{31}$$

This algorithm is similar to the conjugate gradient algorithm—and has similar performance.

7 Iterative Scaling

Iterative scaling is a lower bound method for finding the likelihood maximum. At each step, we construct a simple lower bound to the likelihood and then move to the maximum of this bound. Iterative scaling 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:

$$\begin{aligned} -\log(x) &\geq 1 - \frac{x}{x_0} - \log(x_0) \qquad \text{for any } x_0 && (32) \\ -\exp(-\sum_k q_k w_k) &\geq -\sum_k q_k \exp(-w_k) - (1 - \sum_k q_k) && (33) \\ \text{for any } q_k > 0 \text{ satisfying } \sum_k q_k &\leq 1 \end{aligned}$$

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

$$\begin{aligned} \exp(-\sum_k q_k w_k) &\leq \sum_k q_k \exp(-w_k) && (34) \\ \text{if } \sum_k q_k &= 1 && (35) \end{aligned}$$

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

$$\begin{aligned} \exp(-\sum_k q_k w_k) &\leq \sum_k q_k \exp(-w_k) + (1 - \sum_k q_k) && (36) \\ \text{if } \sum_k q_k &\leq 1 && (37) \end{aligned}$$

Start by writing the likelihood in an asymmetric way:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i|y_i=1} \frac{\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \prod_{i|y_i=-1} \frac{1}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \qquad (38)$$

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

is bounded by

$$\log p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \sum_{i|y_i=1} \mathbf{w}^{\mathrm{T}}\mathbf{x}_i - \sum_i \log(1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) \quad (39)$$

$$\geq \log p(\mathbf{y}|\mathbf{X},\mathbf{w}_0) + \sum_{i|y_i=1} (\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i + \sum_i (1 - \frac{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1+\exp(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i)}) \quad (40)$$

$$= \log p(\mathbf{y}|\mathbf{X},\mathbf{w}_0) + \sum_{i|y_i=1} (\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i + \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)(1-\exp((\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i)) \quad (41)$$

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

$$-\exp((\mathbf{w}-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i) = -\exp(\sum_k (w_k - w_{0k})x_{ik}) \quad (42)$$

$$\geq -\sum_k \frac{x_{ik}}{s}\exp((w_k-w_{0k})s) - (1-\sum_k \frac{x_{ik}}{s}) \quad (43)$$

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

$$g(w_k) = \sum_{i|y_i=1} (w_k - w_{0k})x_{ik} - \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)\sum_k \frac{x_{ik}}{s}\exp((w_k-w_{0k})s) \quad (44)$$

Zero the gradient with respect to w_k:

$$\frac{dg(w_k)}{dw_k} = \sum_{i|y_i=1} x_{ik} - \sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}\exp((w_k-w_{0k})s) = 0 \quad (45)$$

$$\exp((w_k-w_{0k})s) = \frac{\sum_{i|y_i=1} x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}} \quad (46)$$

$$w_k = w_{0k} + \frac{1}{s}\log\frac{\sum_{i|y_i=1} x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}x_i)x_{ik}} \quad (47)$$

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

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \prod_{i|y_i=1} \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \prod_{i|y_i=-1} \frac{\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \quad (48)$$

which would have led to the update

$$\exp((w_k-w_{0k})s) = \frac{\sum_i (1-\sigma(\mathbf{w}_0^{\mathrm{T}}x_i))x_{ik}}{\sum_{i|y_i=-1} x_{ik}} \quad (49)$$

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

$$\exp((w_k - w_{0k})s) = \frac{\sum_{i|y_i=1} x_{ik}}{\sum_{i|y_i=-1} x_{ik}} \frac{\sum_i (1 - \sigma(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i))x_{ik}}{\sum_i \sigma(\mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i)x_{ik}} \tag{50}$$

This is the iterative scaling update used in practice (Nigam et al., 1999; Collins et al., 2002), 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.

8 Modified Iterative Scaling

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

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_i \frac{1}{1 + \exp(-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i)} \tag{51}$$

Applying the first bound (32) at the current parameter values $\mathbf{w}_0$, we obtain that the log-likelihood function is bounded by

$$\begin{aligned}
\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= -\sum_i \log(1 + \exp(-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) && (52)\\
&\geq \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_i (1 - \frac{1 + \exp(-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{1 + \exp(-y_i \mathbf{w}_0^{\mathrm{T}}\mathbf{x}_i)}) && (53)\\
&= \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}_0) + \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i))(1 - \exp(-y_i(\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i)) && (54)
\end{aligned}$$

Maximizing this bound over $\mathbf{w}$ is still too hard. So we apply the second bound (33) with $q_k = x_{ik}/s$, remembering that $x_{ik} > 0$:

$$-\exp(-y_i(\mathbf{w} - \mathbf{w}_0)^{\mathrm{T}}\mathbf{x}_i) \geq -\sum_k \frac{x_{ik}}{s} \exp(-y_i s(w_k - w_{0k})) - (1 - \sum_k \frac{x_{ik}}{s}) \tag{55}$$

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

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

The gradient with respect to w_k is

$$\frac{dg(w_k)}{dw_k} = \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) y_i x_{ik} \exp(-y_i s(w_k - w_{0k})) = 0 \tag{57}$$

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

$$\exp(2s(w_k - w_{0k})) = \frac{\sum_{i|y_i=1} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) x_{ik}}{\sum_{i|y_i=-1} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) x_{ik}} \tag{58}$$

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

$$g(w_k) = -\sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) \sum_k \frac{|x_{ik}|}{s} \exp(-y_i \mathrm{sign}(x_{ik}) s(w_k - w_{0k})) \tag{59}$$

$$\frac{dg(w_k)}{dw_k} = \sum_i (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) y_i x_{ik} \exp(-y_i \mathrm{sign}(x_{ik}) s(w_k - w_{0k})) = 0 \tag{60}$$

$$\exp(2s(w_k - w_{0k})) = \frac{\sum_{i|y_i x_{ik}>0} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) |x_{ik}|}{\sum_{i|y_i x_{ik}<0} (1 - \sigma(y_i \mathbf{w}_0^{\mathrm{T}} x_i)) |x_{ik}|} \tag{61}$$

This update rule was also given by Collins et al. (2002), using a boosting argument. This algorithm will be called Modified Iterative Scaling. The cost is $O(nd)$ per iteration.

9 Dual optimization

The idea behind dual optimization is to replace the original maximization problem with a minimization problem on a completely different function which happens to share the same stationary points. To accomplish this bit of magic, you introduce a set of tight *upper* bounds on $l(\mathbf{w})$ which are parameterized by $\boldsymbol{\alpha}$. The bounds should be simple enough, e.g. quadratic, that the maximum over $\mathbf{w}$ can be computed analytically for each bound. Then the solution to the original problem is given by the upper bound whose maximum over $\mathbf{w}$ has the *smallest* value. Finding this upper bound corresponds to a minimization over $\boldsymbol{\alpha}$, and that is the dual problem.

Jaakkola & Haussler (1999) introduced the first dual algorithm for logistic regression, based on methods used for the Support Vector Machine. The algorithm comes from the following linear upper bound:

$$\log \sigma(x) \leq \alpha x - H(\alpha) \qquad \alpha \in [0, 1] \tag{62}$$

$$\text{where } H(\alpha) = -\alpha \log \alpha - (1 - \alpha) \log(1 - \alpha) \tag{63}$$

Applying this bound throughout $l(\mathbf{w})$ gives an upper bound which is quadratic in $\mathbf{w}$:

$$l(\mathbf{w}) = \sum_i \log \sigma(y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i) - \frac{\lambda}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w} \tag{64}$$

$$\leq l(\mathbf{w}, \boldsymbol{\alpha}) \tag{65}$$

$$\text{where } l(\mathbf{w}, \boldsymbol{\alpha}) = \sum_i \alpha_i y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i - H(\alpha_i) - \frac{\lambda}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w} \tag{66}$$

Now fold in the maximum over $\mathbf{w}$ to get the dual problem:

$$\begin{aligned}
\mathbf{w}(\boldsymbol{\alpha}) &= \lambda^{-1} \sum_i \alpha_i y_i \mathbf{x}_i && (67)\\
J(\boldsymbol{\alpha}) &= \max_{\mathbf{w}} l(\mathbf{w}, \boldsymbol{\alpha}) && (68)\\
&= \frac{1}{2\lambda} \sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i - \sum_i H(\alpha_i) && (69)
\end{aligned}$$

Some things to note about the dual problem:

1. The dimensionality of $\mathbf{w}$ is d, but the dimensionality of $\boldsymbol{\alpha}$ is n. Therefore the dual problem is simpler than the original when the data dimensionality is high ($d > n$).
2. The dual problem only involves inner products between data points. These inner products can be replaced by a Mercer kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ to give a kernelized logistic regression algorithm (Jaakkola & Haussler, 1999). The regularization parameter λ can be absorbed into K.

At this point, we can apply a variety of algorithms to optimize $\boldsymbol{\alpha}$. For example, we could use Newton's method. The derivatives of J are

$$\begin{aligned}
\frac{dJ(\boldsymbol{\alpha})}{d\alpha_i} &= \lambda^{-1} y_i \sum_j \alpha_j y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i + \log \frac{\alpha_i}{1-\alpha_i} && (70)\\
&= y_i \mathbf{w}(\boldsymbol{\alpha})^{\mathrm{T}} \mathbf{x}_i + \log \frac{\alpha_i}{1-\alpha_i} && (71)\\
\frac{d^2 J(\boldsymbol{\alpha})}{d\alpha_i^2} &= \lambda^{-1} \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i + \frac{1}{\alpha_i(1-\alpha_i)} && (72)\\
\frac{d^2 J(\boldsymbol{\alpha})}{d\alpha_i d\alpha_j} &= \lambda^{-1} y_i y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i && (73)
\end{aligned}$$

Unfortunately, Newton's method on the full $\boldsymbol{\alpha}$ vector requires inverting the $n \times n$ Hessian matrix

$$\mathbf{H} = \lambda^{-1} \mathrm{diag}(\mathbf{y}) \mathbf{X}^{\mathrm{T}} \mathbf{X} \mathrm{diag}(\mathbf{y}) + \mathrm{diag}\left(\frac{1}{\alpha_i(1-\alpha_i)} \right) \quad (74)$$

Jaakkola & Haussler (1999) recommend instead to update one α_i at a time, but don't say exactly how. One approach is coordinate-wise Newton:

$$\begin{aligned}
g_i &= y_i \mathbf{w}(\boldsymbol{\alpha})^{\mathrm{T}} \mathbf{x}_i + \log \frac{\alpha_i}{1-\alpha_i} && (75)\\
\alpha_i^{new} &= \alpha_i - \frac{g_i}{\lambda^{-1} \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i + \frac{1}{\alpha_i(1-\alpha_i)}} && (76)
\end{aligned}$$

If this update would take α_i outside the region $[0, 1]$, then we stop it at the endpoint.

The algorithm is efficient when $d > n$. Assuming no kernel is used, 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 α_i's) takes $O(n^2)$ time.

Instead of cycling through all α's, Keerthi et al. (2002) suggest a priority scheme: always update the α_i whose gradient has the largest magnitude. This significantly speeds up the algorithm, because it focuses on the data points near the boundary. To implement this efficiently, the gradients should be incrementally updated. Assuming α_i has changed, the new gradient for α_j is:

$$g_j^{new} = g_j^{old} + \lambda^{-1}(\alpha_i^{new} - \alpha_i^{old}) y_i y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x}_i + \delta(i-j)(\log \frac{\alpha_i^{new}}{1-\alpha_i^{new}} - \log \frac{\alpha_i^{old}}{1-\alpha_i^{old}}) \quad (77)$$

This dual method technically cannot compute an MLE since it requires a proper prior ($\lambda > 0$). One workaround is to make λ very small. However, the convergence rate depends strongly on λ—many iterations are required if λ is small. So the best scheme for an MLE would be to start with large λ and gradually anneal it to zero.

Other algorithms have recently been proposed which also utilize Mercer kernels within logistic regression (Roth, 2001). However, these are not dual algorithms. They rewrite the original objective $l(\mathbf{w})$ in terms of inner products between data points, and then substitute K. The problem remains a maximization over l, not a minimization of J.

10 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 routines in the Lightspeed Matlab toolbox (Minka, 2002). This is more meaningful than comparing the number of iterations or clock time.

The first experiment repeats the setup of Collins et al. (2002). 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. Using this scaling 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).

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. Interestingly, while BFGS asymptotically performs like Newton, in

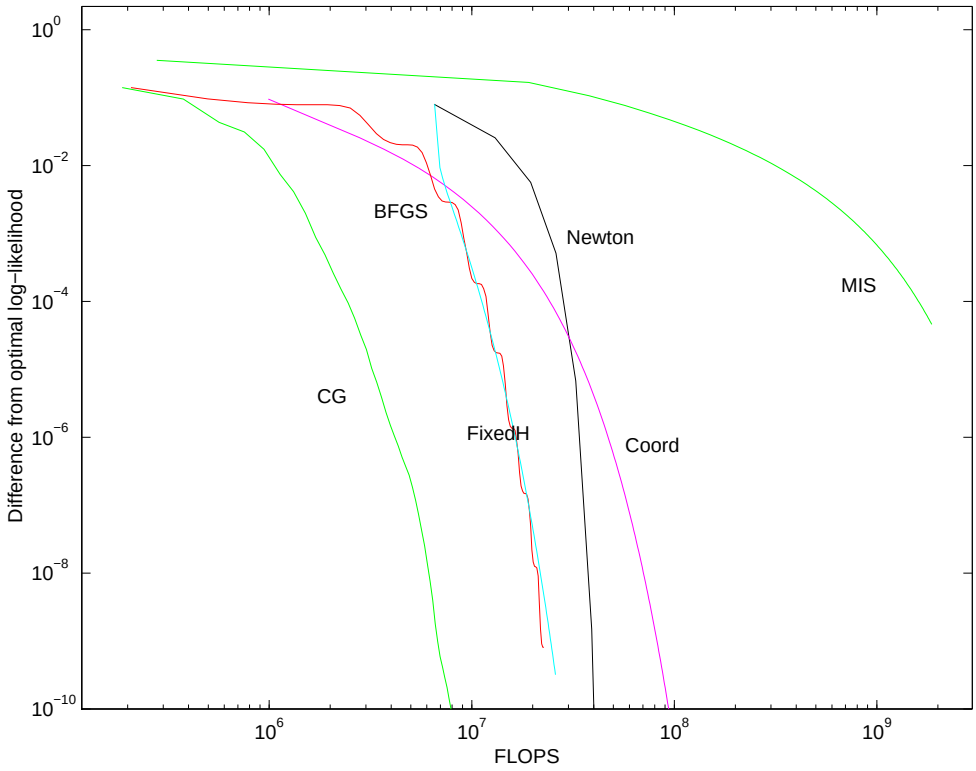

Figure 1: Cost vs. performance of six logistic regression algorithms. The dataset had 300 points in 100 dimensions. "CG" is conjugate gradient (section 4), "Coord" is coordinate-wise Newton, "FixedH" is Fixed-Hessian, and "MIS" is modified iterative scaling. CG also has the lowest actual time in Matlab.

early iterations it seems to behave like Fixed-Hessian. The relative performance of all algorithms remains the same for smaller d, and varies little across repeated draws of the dataset. On bigger problems, such as $(d = 500, n = 1500)$, the differences simply get bigger. The cost difference between CG and MIS is more than three orders of magnitude.

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 c to all feature values x_{ik}. This introduces 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 $w_0 = -c\sum_{i=1}^{d} w_i$. This ensures that $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i$ is unchanged by the shift. Two cases are run: $c = 1$ and $c = 10$. In the latter case, the features are all positive, so Iterative Scaling can be used.

Figures 2 and 3 show the results. As expected, the coordinate-wise algorithms perform poorly, and the Hessian-based algorithms perform best. 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.

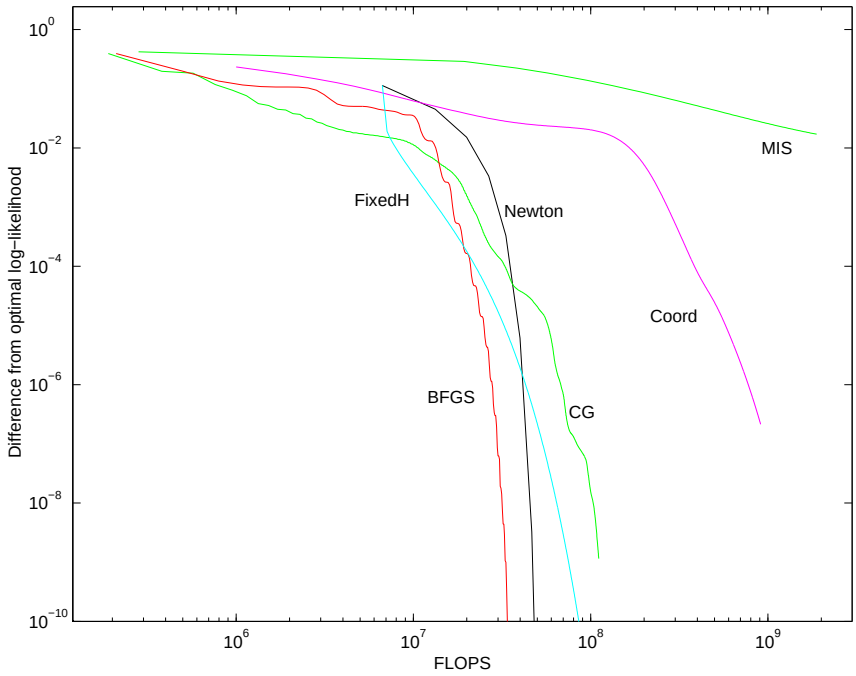

Figure 2: Cost vs. performance of logistic regression algorithms on weakly correlated data. The dataset had 300 points in 100 dimensions.

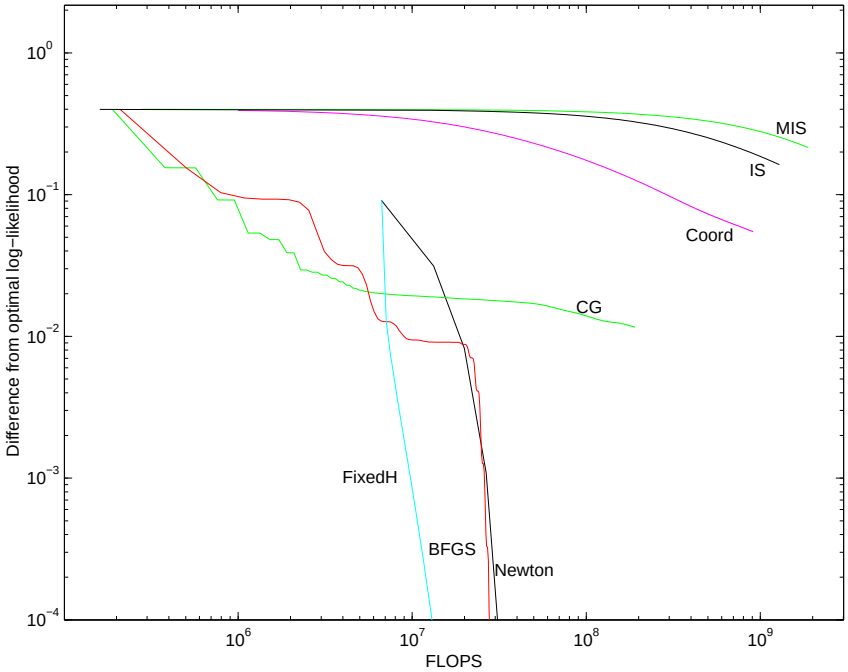

Figure 3: Cost vs. performance of logistic regression algorithms on strongly correlated data. The dataset had 300 points in 100 dimensions.

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

$$p(y=1|\mathbf{x},\mathbf{p},\mathbf{q}) = \frac{\prod_k p_k^{x_{ik}}}{\prod_k p_k^{x_{ik}} + \prod_k q_k^{x_{ik}}} = \sigma(\sum_k x_{ik} \log \frac{p_k}{q_k}) \tag{78}$$

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{79}$$

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.

Figures 4 and 5 show the results for one random dataset of each size. Iterative Scaling performs better than Modified Iterative Scaling and somewhat better than Coordinate-wise Newton, but it is still orders of magnitude behind the leaders.

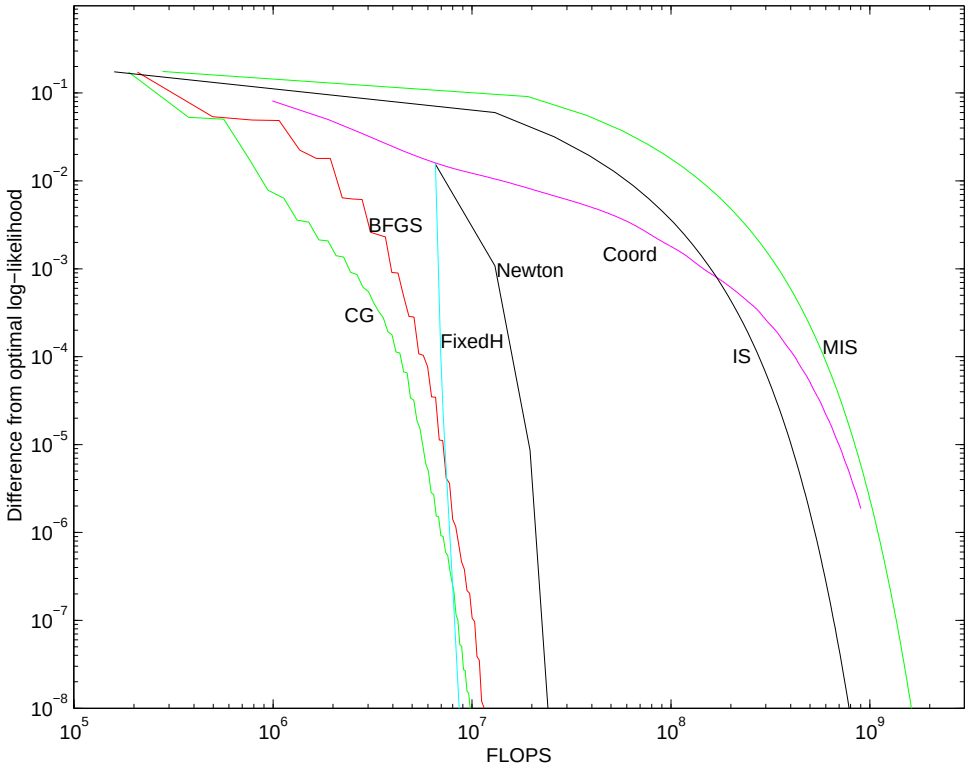

Figure 4: Cost vs. performance of logistic regression algorithms on positive data. The dataset had 300 points in 100 dimensions.

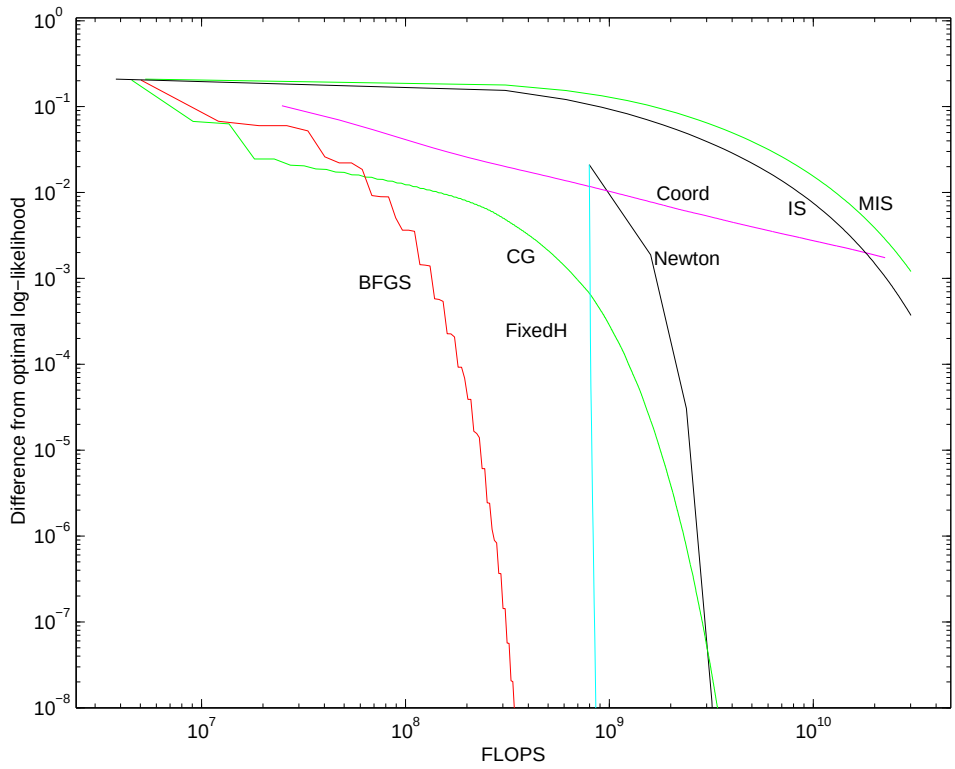

Figure 5: Cost vs. performance of logistic regression algorithms on positive data. The dataset had 1500 points in 500 dimensions.

To test the dual algorithm, we set $\lambda = 0.01$ on this dataset. Note that $\lambda > 0$ generally makes the problem easier, since the surface becomes more quadratic. Figures 6 and 7 show that the dual algorithm (with priority scheme) has quite rapid convergence rate, but for a reasonable level of error such as 10^{-5}, CG and BFGS are still faster.

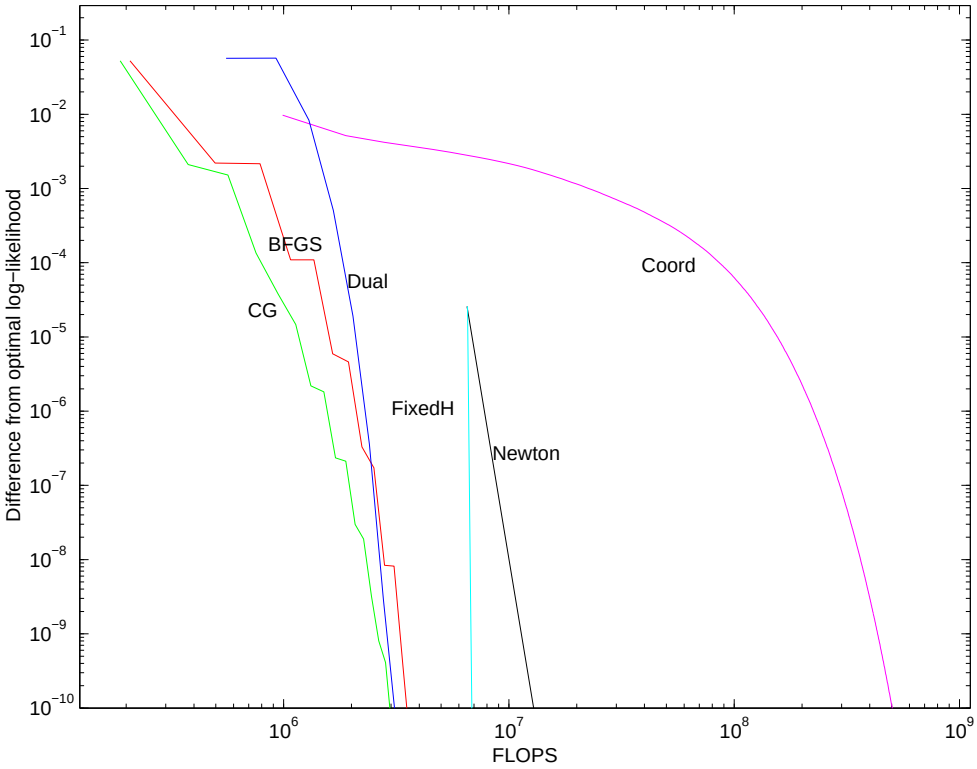

Figure 6: Cost vs. performance of logistic regression algorithms on positive data with $\lambda = 0.01$. The dataset had 300 points in 100 dimensions.

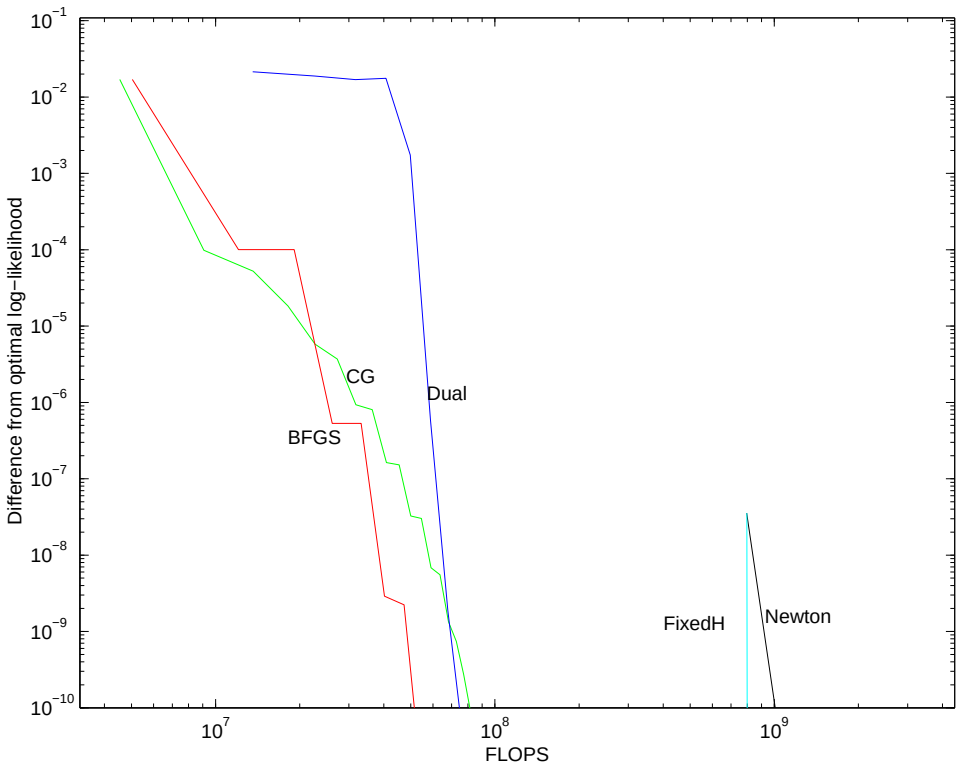

Figure 7: Cost vs. performance of logistic regression algorithms on positive data with $\lambda = 0.01$. The dataset had 1500 points in 500 dimensions.

11 Conclusions

The algorithms can be roughly divided into a Hessian-based group (BFGS, FixedH, Newton) and non-Hessian-based group (CG, Dual, Coord, IS, MIS). In the Hessian-based group, BFGS and FixedH dominate Newton. In the non-Hessian-based group, CG and Dual dominate all others. If the data has strong correlation, then one should use a Hessian-based algorithm. The dimensionality of the data seems to have little impact on the relative performance of the algorithms. All algorithms benefit from using Newton-type line searches.

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. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, *48*, 253–285.
`http://www.cs.princeton.edu/~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`.

Keerthi, S., Duan, K., Shevade, S., & Poo, A. (2002). A fast dual algorithm for kernel logistic regression. *ICML* (pp. 299–306).

Minka, T. (2002). Lightspeed matlab toolbox.
`research.microsoft.com/~minka/software/lightspeed/`.

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/`.

Roth, V. (2001). Probabilistic discriminative kernel classifiers for multi-class problems. *Pattern Recognition–DAGM'01* (pp. 246–253). Springer. LNCS 2191.