Linear Models and Optimization[0]

Xiaoning Qian (xqian@ece.tamu.edu)
ECE, Texas A&M University

ECEN689.611 Machine Learning with Networks

(Chapter 17, A.4-5) January 29, 2014

[0] These slides are modified from the original slides that accompany the book *Bayesian Reasoning and Machine Learning*. The book and demos can be downloaded from `www.cs.ucl.ac.uk/staff/D.Barber/brml`.

The need for optimization

Machine learning often requires fitting a model to data. Often this means finding the parameters $\boldsymbol{\theta}$ of the model that 'best' fit the data.

Regression

For example, for regression based on training data $(\mathbf{x}^n, y^n)$ we might have a model $y(x|\boldsymbol{\theta})$ and wish to set $\boldsymbol{\theta}$ by minimizing

$$\text{Empirical Risk} = E(\theta) = \frac{1}{N}\sum_n \left(y^n - y(\mathbf{x}^n|\theta)\right)^2$$

It also can be derived based on ML estimates by assuming $p(y|\mathbf{x}, \boldsymbol{\theta})$ follows a normal distribution.

Complexity

In all but very simple cases, it is extremely difficult to find an algorithm that will guarantee to find the optimal $\boldsymbol{\theta}$.

Multivariate Calculus

Partial derivative

Consider a function of n variables, $f(x_1, x_2, \ldots, x_n)$ or $f(\mathbf{x})$. The partial derivative of f *w.r.t.* x_i is defined as the following limit (when it exists)

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_n) - f(\mathbf{x})}{h}$$

Gradient vector

For function f the gradient is denoted ∇f or $\mathbf{g}$:

$$\nabla f(\mathbf{x}) \equiv \mathbf{g}(\mathbf{x}) \equiv \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Interpreting the gradient vector

Consider a function $f(\mathbf{x})$ that depends on a vector $\mathbf{x}$. We are interested in how the function changes when the vector $\mathbf{x}$ changes by a small amount : $\mathbf{x} \rightarrow \mathbf{x} + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is a vector whose length is very small:

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + \sum_i \delta_i \frac{\partial f}{\partial x_i} + O\left(\boldsymbol{\delta}^2\right)$$

We can interpret the summation above as the scalar product between the vector ∇f with components $[\nabla f]_i = \frac{\partial f}{\partial x_i}$ and $\boldsymbol{\delta}$.

$$f(\mathbf{x} + \boldsymbol{\delta}) = f(\mathbf{x}) + (\nabla f)^{\mathsf{T}} \boldsymbol{\delta} + O\left(\boldsymbol{\delta}^2\right)$$

Interpreting the Gradient

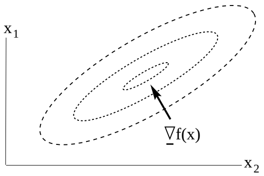


Figure: Interpreting the gradient. The ellipses are contours of constant function value, $f =$ const. At any point $\mathbf{x}$, the gradient vector $\nabla f(\mathbf{x})$ points along the direction of maximal increase of the function.

The gradient points along the direction in which the function increases most rapidly. Why? Consider a direction $\hat{\mathbf{p}}$ (a unit length vector). Then a displacement, δ units along this direction changes the function value to

$$f(\mathbf{x} + \delta\hat{\mathbf{p}}) \approx f(\mathbf{x}) + \delta\nabla f(\mathbf{x}) \cdot \hat{\mathbf{p}}$$

The direction $\hat{\mathbf{p}}$ for which the function has the largest change is that which maximises the overlap

$$\nabla f(\mathbf{x}) \cdot \hat{\mathbf{p}} = |\nabla f(\mathbf{x})||\hat{\mathbf{p}}| \cos\theta = |\nabla f(\mathbf{x})| \cos\theta$$

where θ is the angle between $\hat{\mathbf{p}}$ and $\nabla f(\mathbf{x})$. The overlap is maximised when $\theta = 0$, giving $\hat{\mathbf{p}} = \nabla f(\mathbf{x})/|\nabla f(\mathbf{x})|$. Hence, the direction along which the function changes the most rapidly is along $\nabla f(\mathbf{x})$.

Higher order derivatives

The ‘second-derivative’ of an n-variable function is defined by

$$\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) \qquad i = 1, \ldots, n; \;\; j = 1, \ldots, n$$

which is usually written

$$\frac{\partial^2 f}{\partial x_i \partial x_j}, \;\; i \neq j \qquad \frac{\partial^2 f}{\partial {x_i}^2}, \;\; i = j$$

If the partial derivatives $\partial f/\partial x_i$, $\partial f/\partial x_j$ and $\partial^2 f/\partial x_i \partial x_j$ are continuous, then $\partial^2 f/\partial x_i \partial x_j$ exists and

$$\partial^2 f/\partial x_i \partial x_j = \partial^2 f/\partial x_j \partial x_i \,.$$

This is also denoted by $\nabla\nabla f$. These n^2 second partial derivatives are represented by a square, symmetric matrix called the Hessian matrix of $f(\mathbf{x})$.

$$\mathbf{H}_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{pmatrix}$$

Chain rule

Let each x_j be parameterized by $u_1, \ldots, u_m$, *i.e.* $x_j = x_j(u_1, \ldots, u_m)$.

$$\frac{\partial f}{\partial u_\alpha} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial u_\alpha}$$

or in vector notation

$$\frac{\partial}{\partial u_\alpha} f(\mathbf{x}(\mathbf{u})) = \nabla f^{\mathsf{T}}(\mathbf{x}(\mathbf{u})) \frac{\partial \mathbf{x}(\mathbf{u})}{\partial u_\alpha}$$

Matrix calculus

Gradients of the inner-product:

$$\nabla_{\mathbf{x}} \mathbf{b}^T \mathbf{x} \equiv \mathbf{b}$$

$$\nabla_{\mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{x} \equiv 2\mathbf{A}\mathbf{x}$$

Gradients of the trace:

$$\nabla_{\mathbf{A}} \operatorname{trace}(\mathbf{A}\mathbf{B}) \equiv \mathbf{B}^\mathsf{T}$$

$$\nabla_{\mathbf{A}^\mathsf{T}} \operatorname{trace}\left(\mathbf{A}\mathbf{B}\mathbf{A}^\mathsf{T}\mathbf{C}\right) \equiv \mathbf{B}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{C}^\mathsf{T} + \mathbf{B}\mathbf{A}^\mathsf{T}\mathbf{C}$$

Gradients of the determinant:

$$\nabla_{\mathbf{A}} \det(\mathbf{A}) \equiv \det(\mathbf{A})\, \mathbf{A}^{-\mathsf{T}}$$

$$\nabla_{\mathbf{A}} \log\det(\mathbf{A}) \equiv \mathbf{A}^{-\mathsf{T}}$$

A simple case: Linear regression

For example for a linear predictor

$$y(\mathbf{x}|\boldsymbol{\theta}) \equiv \mathbf{x}^{\mathsf{T}}\boldsymbol{\theta}(= \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}), \text{ or more generally, let } y(\mathbf{x}|\boldsymbol{\theta}) \equiv \boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x})$$

$$E(\boldsymbol{\theta}) = \sum_n \left(y^n - \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n\right)^2$$

The optimum is given when the gradient wrt $\boldsymbol{\theta}$ is zero:

$$\frac{\partial E}{\partial \theta_i} = 2\sum_n \left(y^n - \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n\right) x_i^n = 0$$

$$\underbrace{\sum_n y^n x_i^n}_{b_i} = \sum_j \underbrace{\sum_n x_i^n x_j^n}_{\mathbf{D}_{ij}|\mathbf{D}=\mathbf{X}^{\mathsf{T}}\mathbf{X}} \; \theta_j$$

Hence, in matrix form, this is

$$\mathbf{b} = \mathbf{D}\boldsymbol{\theta}, \rightarrow \boldsymbol{\theta} = \mathbf{D}^{-1}\mathbf{b} = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}$$

which is a simple linear system that can be solved in $O\left((\dim \boldsymbol{\theta})^3\right)$ time.

Quadratic functions

A class of simple functions to optimize is, for symmetric $\mathbf{A}$:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{b}$$

This has a unique minimum if and only if $\mathbf{A}$ is positive definite. In this case, at the optimum

$$\nabla f = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0} \to \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

For $\mathbf{x} + \boldsymbol{\delta}$, the new value of the function is

$$f(\mathbf{x}+\boldsymbol{\delta}) = f(\mathbf{x}) + \underbrace{\boldsymbol{\delta}^\mathsf{T}\nabla f}_{=0} + \frac{1}{2}\underbrace{\boldsymbol{\delta}^\mathsf{T}\mathbf{A}\boldsymbol{\delta}}_{\geq 0}$$

Hence $\mathbf{A}^{-1}\mathbf{b}$ is a minimum.

Linear Models for Classification

Logistic regression

Given a labeled data: $\mathcal{D} = \{\mathbf{x}^n, c^n\}$; learn a linear prediction model for classification: $c(\mathbf{x}^{new})$.
Start from the model for $p(c|\mathbf{x}) = Ber(c|\sigma(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}))$, with its mean function $\sigma(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}) = \frac{e^{\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}{1+e^{\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}} = \frac{1}{1+e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}$.
We would like to maximize the likelihood:

$$\max_{\boldsymbol{\theta}} \Pi_n p(c^n|\mathbf{x}^n).$$

It is equivalent to minimize the negative log likelihood:

$$\min_{\boldsymbol{\theta}} -\sum_n \log p(c^n|\mathbf{x}^n) \rightarrow$$

$$\min_{\boldsymbol{\theta}} -\sum_n c^n \log \sigma(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n) + (1-c^n)\log\left(1-\sigma(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}^n)\right).$$

Gradient Descent

We wish to find $\mathbf{x}$ that minimizes $f(\mathbf{x})$. For general f there is no closed-form solution to this problem and we typically resort to iterative methods.
For $\mathbf{x}_{k+1} \approx \mathbf{x}_k$,

$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}_k) + (\mathbf{x}_{k+1} - \mathbf{x}_k)^\mathsf{T} \nabla f(\mathbf{x}_k)$$

setting

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -\epsilon \nabla f(\mathbf{x}_k)$$

gives

$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}_k) - \epsilon \left|\nabla f(\mathbf{x}_k)\right|^2$$

Hence, for a small ϵ, the algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon \nabla f(\mathbf{x}_k)$$

decreases f. We iterate until convergence.

Gradient Descent for Logistic Regression

For simplicity of representations, we relabel $c' \in \{-1, +1\}$ instead of $c \in \{0, 1\}$ and rewrite the NLL (negative log likelihood):

$$-\sum_n 1(c'^n = 1) \log \sigma(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n) + 1(c'^n = -1) \log (1 - \sigma(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n))$$

$$= \sum_n \log(1 + e^{-c'^n \boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n}).$$

Let $f(\boldsymbol{\theta}) = \sum_n \log(1 + e^{-c'^n \boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n})$, we compute its gradient:

$$\nabla f(\boldsymbol{\theta}) = ??? = \sum_n (\sigma(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n) - c^n)\mathbf{x}^n = \mathbf{X}^\mathsf{T}(\boldsymbol{\sigma} - \mathbf{c}).$$

Hence, implement gradient descent will iterative update $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon_k \nabla f(\boldsymbol{\theta}) = \boldsymbol{\theta}_k + \epsilon_k \sum_n (c^n - \sigma(\boldsymbol{\theta}^\mathsf{T} \mathbf{x}^n))\mathbf{x}^n$ (batch training).

What about "online" training?

Comments on Gradient Descent

Good

This is a very simple algorithm that is easy to implement.

Bad

Only improves the solution at each stage by a small amount. If ϵ is not small enough, the function may not decrease in value. In practice one needs to find a suitably small ϵ to guarantee convergence.

Ugly

The method is coordinate system dependent. Let $\mathbf{x} = \mathbf{M}\mathbf{y}$ and define

$$\hat{f}(\mathbf{y}) = f(\mathbf{x})$$

We now perform gradient descent on $\hat{f}(\mathbf{y})$:

$$[\mathbf{y}_{k+1}]_i = [\mathbf{y}_k]_i - \epsilon \sum_j \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial y_i} \rightarrow \mathbf{y}_{k+1} = \mathbf{y}_k - \epsilon \mathbf{M}^{\mathsf{T}} \nabla f(\mathbf{x}_k)$$

Hence

$$\mathbf{M}\mathbf{y}_{k+1} = \mathbf{M}\mathbf{y}_k - \epsilon \mathbf{M}\mathbf{M}^{\mathsf{T}} \nabla f(\mathbf{x}_k) \rightarrow \mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon \mathbf{M}\mathbf{M}^{\mathsf{T}} \nabla f(\mathbf{x}_k)$$

The algorithm is coordinate system dependent (except for orthogonal $\mathbf{M}$).

Line Search

One way to potentially improve on gradient descent is choose a particular direction $\mathbf{p}_k$ and search along there. We then find the minimum of the one dimensional problem

$$F(\lambda) = f(\mathbf{x}_k + \lambda \mathbf{p}_k)$$

Finding the optimal λ^* can be achieved using a standard one-dimensional optimization method. For example if we identify a bracket such that $f(a) > f(b) < f(c)$ and then fit a polynomial to estimate a new x with $f(x) < f(b)$, this identifies a new bracket. One then repeats until convergence. Once found we set

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda^* \mathbf{p}_k$$

and then choose another search direction $\mathbf{p}_{k+1}$ and iterate.

Good

By reducing the problem to a sequence of one dimensional optimization problems, at each stage the potential change in f is greater than would be typically achievable by gradient descent.

Search directions

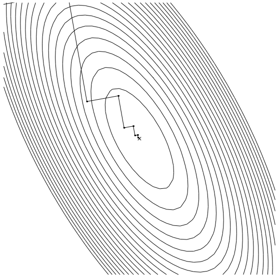
How do we choose a good search direction?

Choosing the search directions

It would seem reasonable to choose a search direction that points ‘maximally downhill’ from the current point $\mathbf{x}_k$. That is, to set

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

However, at least for quadratic functions, this is not optimal and leads to potentially zig-zag behavior: $\nabla F(\lambda) = \mathbf{p}_k^\mathsf{T} \nabla f(\mathbf{x}_{k+1}) = 0$.



This zig-zag behavior occurs for non axis-aligned surfaces. One quick fix is known as heavy ball method: $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon_k \nabla f(\boldsymbol{\theta}_k) + \mu_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$.

Philosophy

Use Quadratic functions to gain insight

Much of the theory of optimization is based on the following: Even though the problem is not quadratic, let's pretend the problem is quadratic and see what we would do in that case. This will inspire the solution for the general case.

Quadratic functions

In the quadratic function case, we get zig-zag behavior if $\mathbf{A}$ is not diagonal. If we could find a coordinate transform $\mathbf{x} = \mathbf{P}\hat{\mathbf{x}}$

$$\hat{f}(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^{\mathsf{T}}\mathbf{P}^{\mathsf{T}}\mathbf{A}\mathbf{P}\hat{\mathbf{x}} - \mathbf{b}^{\mathsf{T}}\mathbf{P}\hat{\mathbf{x}}$$

with

$$\mathbf{P}^{\mathsf{T}}\mathbf{A}\mathbf{P}$$

being diagonal, then we can perform line-search independently along each axis of $\hat{\mathbf{x}}$ and find the solution efficiently. This is achieved by the conjugate gradient algorithm.

Conjugate vectors

The vectors $\mathbf{p}_i$, $i = 1, \ldots, n$ are conjugate for the $n \times n$ matrix $\mathbf{A}$ if

$$\mathbf{p}_i^\mathsf{T} \mathbf{A} \mathbf{p}_j = \delta_{ij} \mathbf{p}_i^\mathsf{T} \mathbf{A} \mathbf{p}_i, \qquad i, j = 1, \ldots, n$$

Searching along these conjugate directions effectively 'diagonalizes' the problem. To keep the argument simple, set $\mathbf{x}_1 = \mathbf{0}$ and in general

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

where the $\mathbf{p}_k$, $k = 1, \ldots, n$ form a conjugate set. Then for

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\mathsf{T} \mathbf{A} \mathbf{x} - \mathbf{x}^\mathsf{T} \mathbf{b}$$

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \frac{1}{2} \mathbf{x}_k^\mathsf{T} \mathbf{A} \mathbf{x}_k + \alpha_k \mathbf{p}_k^\mathsf{T} \mathbf{A} \mathbf{x}_k + f(\alpha_k \mathbf{p}_k)$$

Since $\mathbf{x}_k$ is a linear combination of $\mathbf{p}_1, \ldots, \mathbf{p}_{k-1}$ then

$$\mathbf{p}_k^\mathsf{T} \mathbf{A} \mathbf{x}_k = 0$$

and we can find the optimal α_k by simply minimising $f(\alpha_k \mathbf{p}_k)$, independently of the previous search directions.

Conjugate Gradients

The question now is how to find the conjugate directions. As we saw, we only require that $\mathbf{p}_k$ is conjugate to $\mathbf{p}_1, \ldots, \mathbf{p}_{k-1}$. Defining $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$, and

$$\beta_k = \mathbf{g}_{k+1}^\mathsf{T}\mathbf{g}_{k+1}/(\mathbf{g}_k^\mathsf{T}\mathbf{g}_k)$$

one can show that for the quadratic function at least,

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k$$

is conjugate to all previous $\mathbf{p}_i$, $i = 1, \ldots, k$.
A more common alternative is the Polak-Ribière formula.

$$\beta_k = \frac{\mathbf{g}_{k+1}^\mathsf{T}(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^\mathsf{T}\mathbf{g}_k}$$

For quadratic functions, with $\dim \mathbf{x} = n$, conjugate gradients is guaranteed to find the optimum in n iterations, each iteration taking $O\left(n^2\right)$ operations. In a more general non-quadratic problem, no such guarantee exists.

Conjugate Gradients

This gives rise to the conjugate gradients for minimising a function $f(\mathbf{x})$

```
1:  $k = 1$
2:  Choose $\mathbf{x}_1$.
3:  $\mathbf{p}_1 = -\mathbf{g}_1$
4:  while $\mathbf{g}_k \neq \mathbf{0}$ do
5:      $\alpha_k = \underset{\alpha_k}{\operatorname{argmin}} f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$        ▷ Line Search
6:      $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
7:      $\beta_k := \mathbf{g}_{k+1}^\mathsf{T} \mathbf{g}_{k+1} / (\mathbf{g}_k^\mathsf{T} \mathbf{g}_k)$
8:      $\mathbf{p}_{k+1} := -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k$
9:      $k = k + 1$
10: end while
```

Newton's method

Consider a function $f(\mathbf{x})$ that we wish to find the minimum of. A Taylor expansion up to second order gives

$$f(\mathbf{x}+\boldsymbol{\Delta}) = f(\mathbf{x}) + \boldsymbol{\Delta}^{\mathsf{T}}\nabla f + \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}}\mathbf{H}_f\boldsymbol{\Delta} + O(|\boldsymbol{\Delta}|^3)$$

The matrix $\mathbf{H}_f$ is the Hessian.
Differentiating the right hand side with respect to $\boldsymbol{\Delta}$ (or, equivalently, completing the square), we find that the right hand side has its lowest value when

$$\nabla f = -\mathbf{H}_f\boldsymbol{\Delta} \Rightarrow \boldsymbol{\Delta} = -\mathbf{H}_f^{-1}\nabla f$$

Hence, an optimization routine to minimize f is given by the Newton update

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon\mathbf{H}_f^{-1}\nabla f$$

For quadratic functions, Newton's method converges in one step (for $\epsilon = 1$). More generally one uses a value $\epsilon < 1$ to avoid overshooting effects.

Newton's method for logistic regression

$$H = \nabla\nabla f(\theta) = ??? = \sum_n \sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n)(1-\sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n))\mathbf{x}^n\mathbf{x}^{nT} = \mathbf{X}^{\mathsf{T}}\mathbf{S}\mathbf{X}$$

, where $\mathbf{S} = diag(\sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n)(1-\sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^n)))$.

Iteratively Reweighted Least Squares (IRLS)

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H_k}^{-1}\mathbf{g}_k = ??? = (\mathbf{X}^T\mathbf{S_k}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S_k}\mathbf{z}_k,$$

where $\mathbf{z}_k = \mathbf{X}\boldsymbol{\theta}_k + \mathbf{S_k}^{-1}(\mathbf{c} - Vec(\sigma(\boldsymbol{\theta}_k^\mathsf{T}\mathbf{x}))$.

Comments on Newton's method

Good

A benefit of Newton method over gradient descent is that the decrease in the objective function is invariant under a linear change of co-ordinates, $\mathbf{x} = \mathbf{M}\mathbf{y}$. Defining $\hat{f}(\mathbf{y}) \equiv f(\mathbf{x})$, the change in $\mathbf{y}$ under a Newton update is

$$\mathbf{H}_{\hat{f}}^{-1} \nabla_{\mathbf{y}} \hat{f}$$

where $\nabla_{\mathbf{y}} \hat{f} = \mathbf{M}^{\mathsf{T}} \nabla_{\mathbf{x}} f$, $\mathbf{H}_{\hat{f}} = \mathbf{M}^{\mathsf{T}} \mathbf{H}_f \mathbf{M}$. In terms of the $\mathbf{x}$ coordinate system the change is

$$-\mathbf{M}\Delta\mathbf{y} = -\mathbf{M}\mathbf{H}_{\hat{f}}^{-1} \nabla_{\mathbf{y}} \hat{f} = -\mathbf{M}\left(\mathbf{M}^{\mathsf{T}} \mathbf{H}_f \mathbf{M}\right)^{-1} \mathbf{M}^{\mathsf{T}} \nabla_{\mathbf{x}} f = -\mathbf{H}_f^{-1} \nabla_{\mathbf{x}} f$$

so that the change is independent of the coordinate system (up to linear transformations of the coordinates).

Bad

Storing the Hessian and solving the linear system $\mathbf{H}_f^{-1} \nabla f$ is very expensive.

Quasi-Newton

In Quasi-Newton methods such as Broyden-Fletcher-Goldfarb-Shanno, an approximate inverse Hessian is formed iteratively.