CS5014 Machine Learning

Lecture 6 Logistic regression and Newton's method

Lei Fang

School of Computer Science, University of St Andrews

Spring 2021

University
of
St Andrews

Poll last time

1. How do you think about the level of difficulty of maths ?

More Details

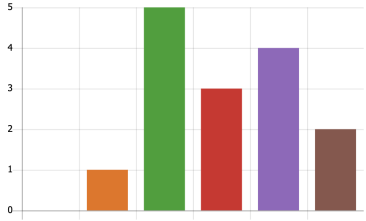
too easy	0
easy to follow	3
challenging but manageable	11
too hard	1

2. Speed of delivery?

More Details

View all responses for question 2

Too slow	0
can be faster	1
good	5
can be slower	3
Too fast	4
Other	2



Maths

Do I need to read all the maths on the slides *during lecture* ?

- **NO**! follow the logic is more important
- *esp.* when I jump slides (most likely technical details)
  - e.g. "take the derivative and set it to zero", just believe me for the time being
  - might left there for your reference (I will make sure it is clear)
- catch the key message
  - the conclusion: e.g. *MLE* leads to least square

BUT **YES** verify them after the lecture step by step

- no way to learn CS or maths modules just by attending lectures

Feel free to stop me during the lecture

- when you cannot follow the logic: highly likely I have messed something up
- when the notation is confusing: still possibly my bad
  - I used σ for both sigmoid and variance (σ^2) of Gaussian
  - or like $\prod p_i^{I(x=i)}$, if you do not know what I does..

Do I need to do the derivations?

Yes and no.

- No; not a maths module *per se*
- Yes; to get a 20/20, we expect you to
  - know the details
  - solve noval problems (derive algorithms based on some unseen models)

Take gradients for example

- not a big deal if you mess up a gradient derivation
- but **very problematic** if you *after taking this module*
  - do not know what gradient does
  - do not know e.g. $f : R^n \rightarrow R$'s derivative is a n-element vector
  - do not know gradient is actually a function (it returns a direction given a "location")
- less problematic if you
  - do not know how to solve the problem if you mess it up (debug)

Speed of delivery

- I will slow down a bit
- will not cut down any material though
- cover as much as we can in one hour
- provide extra video clips for missing slides
  - or parts need more examples
  - polls at the end

Recap

Maximum likelihood estimation (MLE):

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}} \underbrace{\log P(\mathcal{D}|\theta)}_{\mathcal{L}(\theta)}$$

- $\mathcal{L}(\theta)$: log likelihood function
- θ_{ML}: maximum likelihood estimator

Two key results

- Gaussian likelihood $\Leftrightarrow$ Loss for regression (squared error loss)
- Bernoulli likelihood $\Leftrightarrow$ Loss for binary classification (cross-entropy loss)

The moral of the story:

(negative) Log likelihoods are loss functions (with some nice properties)

Topics of today

Logistic regression

- closer look at the loss function
- gradients of logistic regression
- stochastic gradient descent

Newton's method

- another optimisation algorithm
- converge faster than gradient descent (theoretically)

MLE's asymptotic distribution (pre-recorded video clip)

- θ_{ML} is actually a random variable
- $\theta_{ML} \sim N(\theta, I_n^{-1})$

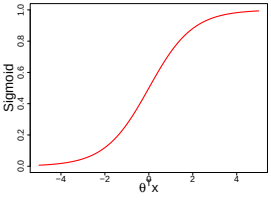
Logistic regression

Classification problem: $y^{(i)} \in \{0, 1\}$, predictors $\boldsymbol{x}^{(i)} \in R^n$
The model:

$$P(y^{(i)} = 1|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}), \quad P(y^{(i)} = 0|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) = 1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$$

which can be written as (why?)

$$P(y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) = \sigma(\cdot)^{y^{(i)}}(1 - \sigma(\cdot))^{1-y^{(i)}}.$$



The log likelihood function is

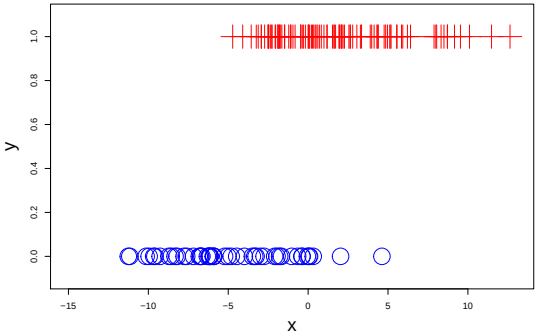
$$\mathcal{L}(\boldsymbol{\theta}) = \log \prod_{i=1}^{m} P(y^{(i)}|\boldsymbol{\theta}, \boldsymbol{x}^{(i)}) = \sum_{i=1}^{m} y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)})$$

shorthand notation: $\sigma^{(i)} \equiv \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$; superscript with brackets, (i):
index of observations

Example

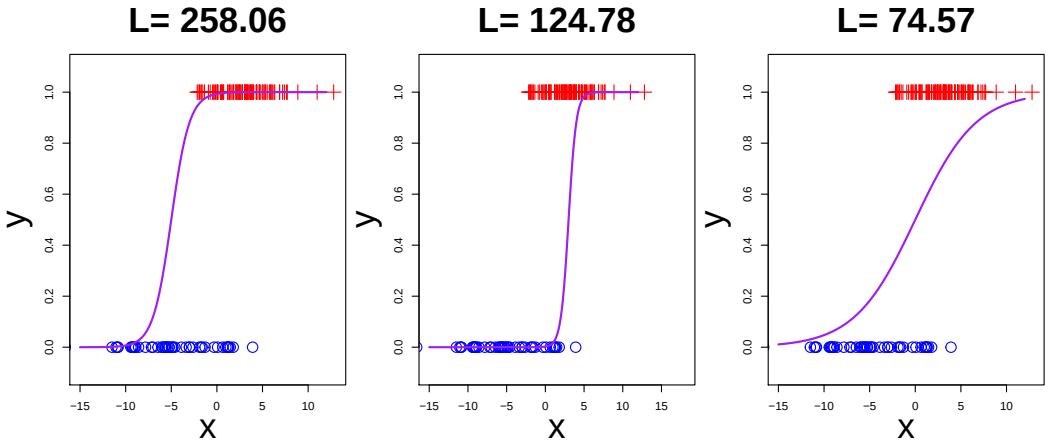
Let's consider a 1-d example

- blue 'o' are negative: $y^{(i)} = 0$
- red '+' are positive: $y^{(i)} = 1$
- with $P(y^{(i)} = 1) = \sigma(\theta_1 x^{(i)} + \theta_0)$ in mind
- learning: how to draw such a *sigmoid* curve?



The negative log-likelihood $-\mathcal{L}$ provides a measure of fit

- maximise $\mathcal{L}$ is the same as minimise $-\mathcal{L}$
- loss function for classification
- smaller the better



Closer look at the loss

Consider one data point i: $y^{(i)}, \boldsymbol{x}^{(i)}$

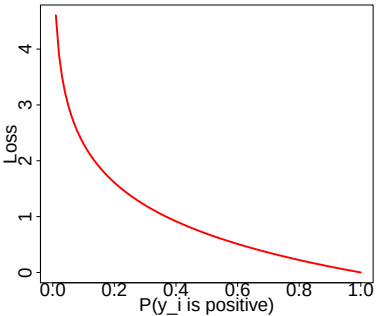
- the negative log likelihood is

$$-l^{(i)} = y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)})$$

- consider $y^{(i)} = 1$, *i.e.* positive case; we want the probability

$$P(y^{(i)} = 1) = \sigma^{(i)}$$

big (or close to 1)



Even closer look at the loss

Exponential cost/lost (left side of the red curve)

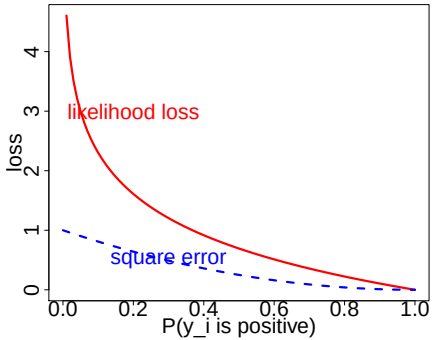
- penalise confident but wrong answers
- is it a good thing ?
  - it kinda makes sense
  - but also not: outliers ?
  - heavy influence by one data on the final model

Flat curve on the right side

- $P = 0.8, 0.9, 1.0$ the losses are similar
- discourage over confident predictions

You can use squared error actually

- called "Brier score" loss
- can you tell any potential problem ?



Gradient descent of logistic regression

Let's talk about fitting that "S" shaped surface

Objective: maximising the log likelihood

$$\begin{aligned}\boldsymbol{\theta}_{ML} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}(-\mathcal{L}(\boldsymbol{\theta})) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \left(\sum_{i=1}^{m} y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)}) \right)\end{aligned}$$

- there is no closed-form solution to: $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$
  - $\mathcal{L}$ is a nonlinear transformation of $\boldsymbol{\theta}$
  - but $-\mathcal{L}$ still convex: only one minimum
- you should know what algorithm to use now
  - gradient descent

Gradient descent

Let's do some derivative:

$$\mathcal{L}(\boldsymbol{\theta}) = \left(\sum_{i=1}^{m} y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)}) \right)$$

Note that $\sigma^{(i)} = \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)})$
Let's consider ith summand only:

$$l^{(i)}(\boldsymbol{\theta}) = y^{(i)} \log \sigma^{(i)} + (1 - y^{(i)}) \log(1 - \sigma^{(i)})$$

$$\nabla_{\boldsymbol{\theta}} l^{(i)} = ?$$

The following identities might be useful: $\frac{d\sigma(x)}{dx} = \sigma(1 - \sigma)$ and $\frac{\partial \boldsymbol{a}^T \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a}^T$

Gradient of logistic regression

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} l^{(i)} &= \left(y^{(i)} - \sigma^{(i)}\right)\left(\boldsymbol{x}^{(i)}\right)^T \\
\nabla_{\boldsymbol{\theta}} \mathcal{L} &= \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} l^{(i)} \\
&= \sum_{i=1}^{m} \left(y^{(i)} - \sigma^{(i)}\right)\left(\boldsymbol{x}^{(i)}\right)^T \\
&= \left(\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta})\right)^T \boldsymbol{X}
\end{aligned}
$$

I am abusing the notation here: σ is $R \to R$, cannot be applied to a vector input. It means apply σ to each element of $\boldsymbol{X}\boldsymbol{\theta}$

Gradient of logistic regression

$$\nabla_{\boldsymbol{\theta}} l^{(i)} = \left(y^{(i)} - \sigma^{(i)}\right)\left(\boldsymbol{x}^{(i)}\right)^T$$

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{L} &= \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} l^{(i)} \\ &= \sum_{i=1}^{m} \left(y^{(i)} - \sigma^{(i)}\right)\left(\boldsymbol{x}^{(i)}\right)^T \\ &= \left(\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta})\right)^T \boldsymbol{X}\end{aligned}$$

initialise random $\boldsymbol{\theta}_0$;
while *not converge* **do**
  $\boldsymbol{g}_t \leftarrow (-\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t))^T$;
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \boldsymbol{g}_t$;
end

Algorithm 1: Gradient descent

note that $\boldsymbol{g}_t$ is the gradient of negative $\mathcal{L}$, so it is a descending algorithm; also we transpose the gradient (assumed row vector) to a column vector here to make the notation consistent as $\boldsymbol{g}_t$ is assumed a column vector.

Connection to linear regression's gradient

You should find the logistic gradient **familiar**

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}))^T \boldsymbol{X}$$

Remember the gradient for linear regression ?

$$\nabla_{\boldsymbol{\theta}} L = -2(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T \boldsymbol{X}$$

- "2" doesn't matter: L could've been scaled to $\frac{1}{2}L$
- "-" sign because we are maximising $\mathcal{L}$

Essentially it replaces the predictions

$$\boldsymbol{X}\boldsymbol{\theta} = [\boldsymbol{\theta}^T \boldsymbol{x}^{(1)}, \boldsymbol{\theta}^T \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{\theta}^T \boldsymbol{x}^{(m)}]^T$$

with the σ transformed predictions

$$\sigma(\boldsymbol{X}\boldsymbol{\theta}) = [\sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(1)}), \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(1)}), \ldots, \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(m)})]^T$$

*No coincidence: all Generalised Linear Models (GLMs) have the same gradient form

Reminder for vector notation

$\boldsymbol{x}$ by default a column vector, therefore $\boldsymbol{x}^T$ means **row** vector

However, when I write out the elements of $\boldsymbol{x}$, one needs to write

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

which takes too much space in text, so you will usually see

$$\boldsymbol{x} = \overbrace{\underbrace{[x_1, x_2, \ldots, x_n]}_{\text{row vector}}^T}^{\text{column vector}}$$

Just to tell you $\boldsymbol{x}$ is still a column vector

Stochastic gradient descent

When data size is massive, gradients are expensive to calculate

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} l^{(i)}$$

- need to calculate m gradients $\nabla_{\boldsymbol{\theta}} l^{(i)}$
- or not enough memory
- vector addition: adding directions together
- SGD uses one of those directions instead
- no longer steepest descent direction
  - sometimes might ascent
- still guarantee to converge

initialise random θ_0;
while *not converge* **do**
  randomly permute data;
  for $i \in 1 \dots m$ **do**
    $\boldsymbol{g}_t \leftarrow (-\nabla_{\boldsymbol{\theta}} l^{(i)}(\boldsymbol{\theta}_t))^T$;
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \boldsymbol{g}_t$;
  end
end

Algorithm 2: Stochastic gradient descent

Stochastic gradient descent

When data size is massive, gradients are expensive to calculate

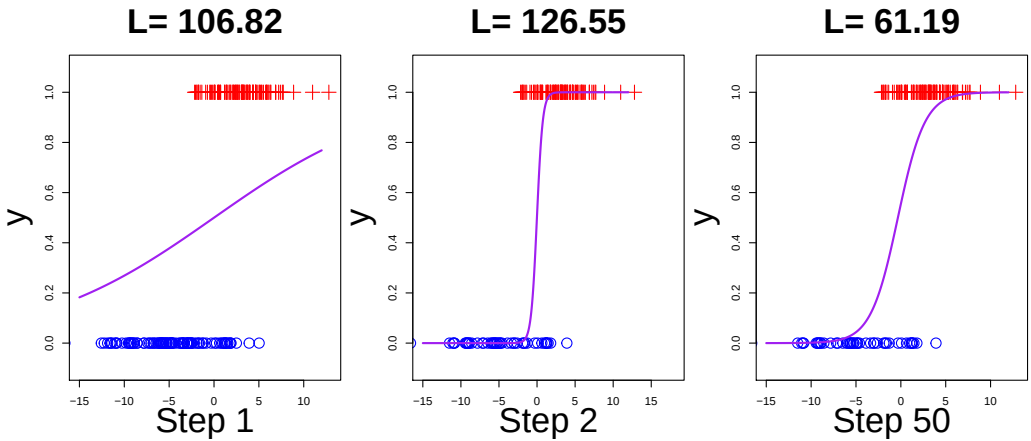
$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} l^{(i)}$$

- need to calculate m gradients $\nabla_{\boldsymbol{\theta}} l^{(i)}$
- or not enough memory
- vector addition: adding directions together
- SGD uses one of those directions instead
- no longer steepest descent direction
  - sometimes might ascent
- still guarantee to converge

initialise random $\boldsymbol{\theta}_0$;
while *not converge* **do**
  randomly permute data;
  for $i \in 1 \dots m$ **do**
    $\boldsymbol{g}_t \leftarrow (-\nabla_{\boldsymbol{\theta}} l^{(i)}(\boldsymbol{\theta}_t))^T$;
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \boldsymbol{g}_t$;
  end
end

Algorithm 3: Stochastic gradient descent

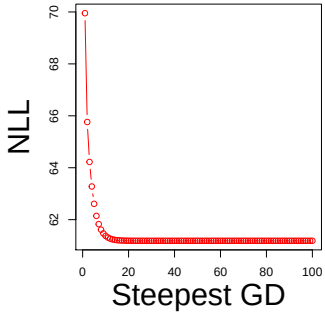
Example gradient descent steps for logistic regression

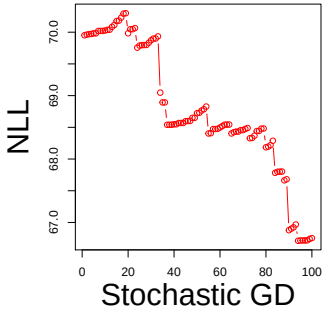


Example: GD vs SGD

Compare Steepest Gradient Descent (GD) and Stochastic Gradient Descent (SGD)

- note that SGD can go up
- every step is no longer descent direction



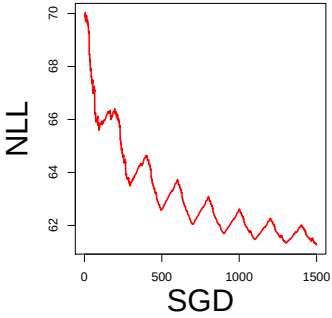


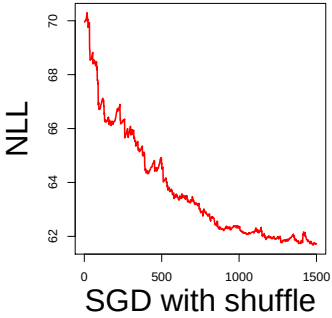
NLL means negative log likelihood

Example: random shuffle SGD

Note that both converge at the end: < 68 at the end like steepest GD

The nonshuffled SGD is pretty cyclic

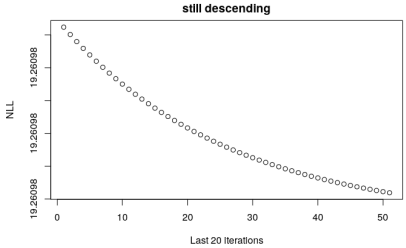




Newton's method: motivation

Gradient descent usually converges very slow at the end

- why though ?
- what the gradient behaves like at the end ? destined to vanish



Solution: intuitively, we need to use the second derivative information

- second derivative: measures curvature
- if it is flat (small curvature), take larger steps
- otherwise, take smaller steps

Newton's method: algorithm in words

Newton's method does exactly that

- approximate L by a quadratic function $T(\theta) = a\theta^2 + b\theta + c$ or

$$T(\theta) = \underbrace{\boldsymbol{\theta}^T \boldsymbol{A} \boldsymbol{\theta}}_{\text{what is this?}} + \boldsymbol{b}^T \boldsymbol{\theta} + c$$

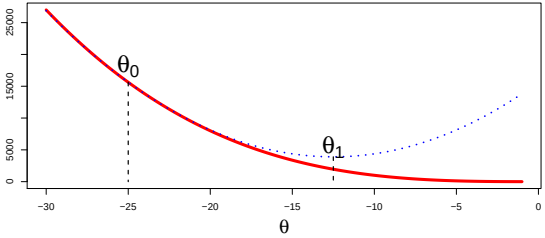
  - why quadratic ?
  - simpliest function with non-zero curvature
  - (hyperplanes have 0 second derivative)
- and optimise that approximation T instead
  - good news: quadratic functions have closed form solutions!

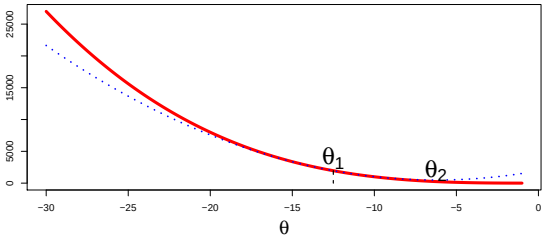
$$\nabla_{\boldsymbol{\theta}} T = 2\boldsymbol{\theta}^T \boldsymbol{A} + \boldsymbol{b}^T = \mathbf{0} \Rightarrow \boldsymbol{\theta}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$$

  - set the new location to $\boldsymbol{\theta}^*$
- and repeat the above two steps until converge

$\boldsymbol{A}$ is assumed symmetric; it turns out all quadratic forms with asymmetric matricies can be massaged to this form

An example: Newton steps





Taylor's expansion

So how to find the approximation? Taylor's expansion:

$$T(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_t) + \underbrace{\boldsymbol{g}_t^T(\boldsymbol{\theta} - \boldsymbol{\theta}_t)}_{\text{linear}} + \frac{1}{2!}\underbrace{(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^T \boldsymbol{H}_t(\boldsymbol{\theta} - \boldsymbol{\theta}_t)}_{\text{quadratic}} + \ldots,$$

- where $\boldsymbol{g}_t = (\nabla_\theta f(\boldsymbol{\theta}_t))^T$ is the gradient of f at $\boldsymbol{\theta}_t$
- $\boldsymbol{H}$ is the Hessian matrix of f: second order derivative at $\boldsymbol{\theta}_t$
- $T(\boldsymbol{\theta})$ (a quadratic function) approximates $f(\boldsymbol{\theta})$ at $\boldsymbol{\theta}_t$: exactly what we want

Last step: we need to optimise T to find the next location $\boldsymbol{\theta}_{t+1}$

$$\nabla_\theta T = \boldsymbol{g}_t^T + (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^T \boldsymbol{H}_t = \mathbf{0}$$

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_t - \boldsymbol{H}_t^{-1}\boldsymbol{g}_t$$

Newton's direction $\boldsymbol{d}_t$ vs gradient direction $\boldsymbol{g}_t$

This guy looks too familar to ignore

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_t - \underbrace{\boldsymbol{H}_t^{-1}\boldsymbol{g}_t}_{\boldsymbol{d}_t}$$

Remember gradient descent step

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_t - \alpha \boldsymbol{g}_t$$

- $\boldsymbol{d}_t = \boldsymbol{H}_t^{-1}\boldsymbol{g}_t$ is a transformed gradient direction: Newton's direction
- if $\boldsymbol{H}_t^{-1} = \boldsymbol{I}$, you recover the same update rule; can you see why ?
- what if $\boldsymbol{H}_t$ is diagonal with different entries ? *e.g.* $\boldsymbol{H}_t = \begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}$

Moral of the story ? The steepest way now may not be optimal at the end. Maths is very philological indeed.

initialise random $\boldsymbol{\theta}_0$;
while *not converge* **do**
  $\boldsymbol{g}_t \leftarrow (-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t))^T$;
  $\boldsymbol{H}_t \leftarrow -\nabla^2_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$;
  solve for Newton's direction: $\boldsymbol{H}_t \boldsymbol{d}_t = \boldsymbol{g}_t$;
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{d}_t$;
end

Algorithm 4: Newton's Method

- direct inverting $\boldsymbol{H}_t$ is expensive
- less so if solve $\boldsymbol{H}_t \boldsymbol{d}_t = \boldsymbol{g}_t$
  - people call this Newton Conjugate Gradient method (Newton CG) (when conjugate gradient is used to solve the system of equations, and we will not discuss conjugate gradient method in this course).

Newton's method for logistic regression

We need to find the Hessian $\boldsymbol{H}$ for the logistic log likelihood

$$\begin{aligned}\boldsymbol{H} &= \nabla^2_{\theta}\mathcal{L} \\ &= \boldsymbol{X}^T\boldsymbol{D}\boldsymbol{X},\end{aligned}$$

where

$$\boldsymbol{D} = \begin{bmatrix} \sigma^{(1)}(\sigma^{(1)}-1) & 0 & \dots & 0 \\ 0 & \sigma^{(2)}(\sigma^{(2)}-1) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^{(m)}(\sigma^{(m)}-1) \end{bmatrix}$$

Together with the gradient, we have a Newton step.

Next time

Practical issues on gradient and Newton like methods

- tips on deriving gradients
- how to spot errors
- compare and contrast

Non-linear models and regularisation

Bonus slides: Hessian for the likelihood of logistic regression

Firstly, for a general function $f(x) : R^n \to R$, the Hessian matrix of f is defined as

$$\nabla_x^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- one key observation: it is symmetric as $\frac{\partial^2 f}{\partial x_i x_j} = \frac{\partial^2 f}{\partial x_j x_i}$

Example:

$$f(\boldsymbol{x}) = 2x_1^2 + 4x_2^2$$

It's gradient is $\nabla_{\boldsymbol{x}} f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}] = [4x_1, 8x_2]$; and by definition its Hessian is

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial f^2}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_x \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 8 \end{bmatrix}$$

Like most maths concepts, definitions are fundamentally important but not very useful in practice.

- e.g. we don't calculate derivative by its limit definition

A more useful identity for Hessian is: it's a gradient of the gradient.

- like what we do in univariate calculus: find the derivative first and take the derivative again to find the second derivative.
- so find $\nabla_x f$ first, then find the gradient of the gradient: $\nabla_x(\nabla_x f)$.
- the process involves applying the chain rule usually.

Example:

$$f(\boldsymbol{x}) = 2x_1^2 + 4x_2^2 = \boldsymbol{x}^T\boldsymbol{M}\boldsymbol{x}, \text{ where } \boldsymbol{x} = [x_1, x_2]^T \quad \boldsymbol{M} = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix},$$

It's gradient is $\nabla_x f = 2\boldsymbol{x}^T\boldsymbol{M}$;
Take derivative again: $\nabla_x(\nabla_x f) = 2\boldsymbol{M}^T = 2\boldsymbol{M}$, as $\boldsymbol{M}$ is symmetric. The results are the same but much simpler to obtain by using matrix derivative identities.
We have used the following identities (useful identities can be found online, say wikipedia):

$$\frac{\partial \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} = 2\boldsymbol{x}^T\boldsymbol{A}; \quad \frac{\partial \boldsymbol{x}^T\boldsymbol{A}}{\partial \boldsymbol{x}} = \boldsymbol{A}^T$$

Now let's tackle the Hessian for logistic regression's log likelihood
Remember the gradient is

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}))^T \boldsymbol{X}$$

The useful matrix derivative identities for the calculation are listed below as a reference

$$\frac{\partial \boldsymbol{x}^T \boldsymbol{A}}{\partial \boldsymbol{x}} = \boldsymbol{A}^T; \quad \frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{A}.$$

To find the Hessian, take the gradient of $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ again

$$\begin{aligned}
\boldsymbol{H} &= \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} \left((\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}))^T \boldsymbol{X} \right) \\
&= \underbrace{\nabla_{(\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}))} \left((\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}))^T \boldsymbol{X} \right)}_{X^T} \cdot \underbrace{\nabla_{(\boldsymbol{X}\boldsymbol{\theta})} \left(\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\theta}) \right)}_{D} \cdot \underbrace{(\nabla_{\boldsymbol{\theta}} \boldsymbol{X}\boldsymbol{\theta})}_{X} \\
&= \boldsymbol{X}^T \boldsymbol{D} \boldsymbol{X}
\end{aligned}$$

The second line has used the chain rule. The first term has used the first identity above; and the third term has used the second identity. You should convince yourself why.
The middle term is a Jacobian matrix actually, as the function transforms a vector to another vector (the gradient of a vector to vector function is a Jacobian matrix). Let's take a closer look.

Firstly, to make the notation less clutter, Let's denote $\boldsymbol{X\theta} \equiv \phi$. Note that $\nabla_{(\boldsymbol{X\theta})} (\boldsymbol{y} - \sigma(\boldsymbol{X\theta})) \equiv \nabla_{\phi} (\boldsymbol{y} - \sigma(\phi)) = -\nabla_{\phi}\sigma(\phi)$ as $\boldsymbol{y}$ is a constant vector w.r.t the input $\boldsymbol{X\theta}$. The function we are considering here transforms ϕ to $\sigma(\phi)$. The input is a $m \times 1$ (m is number of data observations) element vector:

$$\phi \equiv \boldsymbol{X\theta} = [\boldsymbol{\theta}^T \boldsymbol{x}^{(1)}, \boldsymbol{\theta}^T \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{\theta}^T \boldsymbol{x}^{(m)}]^T$$

The output of the function is a $m \times 1$ vector as well:

$$\sigma(\phi) \equiv \sigma(\boldsymbol{X\theta}) = [\sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(1)}), \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(2)}), \ldots, \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(m)})]^T = [\sigma^{(1)}, \sigma^{(2)}, \ldots, \sigma^{(m)}]^T$$

Indeed it is $R^m \to R^m$ functions (vector to vector).
Note that if we consider each scalar of the output vector, say $\sigma^{(i)}$ it is a $R^m \to R$ function.
E.g. the first entry of the output is a function with input ϕ and output $\sigma^{(1)}$ as

$$\sigma^{(1)}(\phi) = 1 \times \sigma(\phi_1) + 0 \times \sigma(\phi_2) + \ldots + 0 \times \sigma(\phi_m) = \sigma(\phi_1)$$

is a vector to scalar function where the input is $\phi \in R^n$ and the output is a scalar. And more generally, the j-th output is

$$\sigma^{(j)}(\phi) = 0 \times \sigma(\phi_1) + \ldots + 1 \times \sigma(\phi_j) + 0 \times \sigma(\phi_m) = \sigma(\phi_j).$$

Then it is easy to find the gradient of the j-th output entry w.r.t ϕ:

$$\nabla_{\phi}\sigma^{(j)} = \left[\frac{\partial\sigma^{(j)}}{\phi_1}, \frac{\partial\sigma^{(j)}}{\phi_2}, \ldots, \frac{\partial\sigma^{(j)}}{\phi_m}\right] = [0 \ldots, \sigma^{(j)}(1-\sigma^{(j)}), \ldots 0],$$

which has only the j-th entry filled with its derivative and all the others 0 (this should be obvious if you plug in the definition of $\sigma^{(j)}(\phi)$ from the previous step).
The Jacobian is a matrix with its rows filled with those individual gradients $\nabla_{\phi}\sigma^{(j)}$:

$$\nabla_{\phi}\sigma(\phi) = \begin{bmatrix} \nabla_{\phi}\sigma^{(1)} \\ \nabla_{\phi}\sigma^{(2)} \\ \vdots \\ \nabla_{\phi}\sigma^{(m)} \end{bmatrix} = \begin{bmatrix} \sigma^{(1)}(1-\sigma^{(1)}) & 0 & \ldots & 0 \\ 0 & \sigma^{(2)}(1-\sigma^{(2)}) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma^{(m)}(1-\sigma^{(m)}) \end{bmatrix},$$

To put everything together, we have $\nabla_{(\boldsymbol{X\theta})}\left(\boldsymbol{y} - \sigma(\boldsymbol{X\theta})\right) = -\nabla_{\phi}\sigma(\phi) = \boldsymbol{D}$.
It worths remembering the following fact: for a general vector to vector function $f : R^n \to R^m$, the Jacobian matrix is $m \times n$ matrix: as we are stacking m rows together (each row for each output entry's gradient) and each row has n entries (as each gradient has n elements).