Lecture 4. Iterative Optimisation & Logistic Regression

COMP90051 Statistical Machine Learning

Semester 1, 2021
Lecturer: Trevor Cohn

THE UNIVERSITY OF MELBOURNE

This lecture

- Iterative optimisation
  - First-order method: Gradient descent
  - Second-order: Newton-Raphson method
  - Later: Lagrangian duality
- Logistic regression: workhorse linear classifier
  - Possibly familiar derivation: frequentist
  - Decision-theoretic derivation

Gradient Descent

Brief review of most basic optimisation approach in ML

Optimisation formulations in ML

- Training = Fitting = Parameter estimation
- Typical formulation

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} L(data, \boldsymbol{\theta})$$

  * argmin because we want a minimiser not the minimum
    - Note: argmin can return a set (minimiser not always unique!)
  * Θ denotes a model family (including constraints)
  * L denotes some objective function to be optimised
    - E.g. MLE: (conditional) likelihood
    - E.g. Decision theory: (regularised) empirical risk

One we've seen: Log trick

- Instead of optimising $L(\theta)$, try convenient $\log L(\theta)$
- Why are we allowed to do this?
- Strictly monotonic function: a $> b \Longrightarrow f(a) > f(b)$
  * Example: log function!
- **Lemma**: Consider any objective function $L(\theta)$ and any strictly monotonic f. θ^* is an optimiser of $L(\theta)$ if and only if it is an optimiser of $f(L(\theta))$.
  * Proof: Try it at home for fun!

Two solution approaches

- Analytic (*aka* closed form) solution
  - Known only in limited number of cases
  - Use 1st-order necessary condition for optimality*:

$$\frac{\partial L}{\partial \theta_1} = \cdots = \frac{\partial L}{\partial \theta_p} = 0$$

Assuming unconstrained, differentiable *L*

- Approximate iterative solution
  1. Initialisation: choose starting guess $\boldsymbol{\theta}^{(1)}$, set $i = 1$
  2. Update: $\boldsymbol{\theta}^{(i+1)} \leftarrow SomeRule\left[\boldsymbol{\theta}^{(i)}\right]$, set $i \leftarrow i + 1$
  3. Termination: decide whether to Stop
  4. Go to Step 2
  5. Stop: return $\widehat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^{(i)}$

* Note: to check for local minimum, need positive 2nd derivative (or Hessian positive definite); this assumes unconstrained – in general need to also check boundaries. See also Lagrangian techniques later in subject.

Reminder: The gradient

- Gradient at $\boldsymbol{\theta}$ defined as $\left[\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_p}\right]'$ evaluated at $\boldsymbol{\theta}$
- The gradient points to the direction of maximal change of $L(\boldsymbol{\theta})$ when departing from point $\boldsymbol{\theta}$
- Shorthand notation
  * $\nabla L \stackrel{\text{def}}{=} \left[\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_p}\right]'$ computed at point $\boldsymbol{\theta}$
  * Here ∇ is the "nabla" symbol
- Hessian matrix at $\boldsymbol{\theta}$: $\nabla^2 L_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}$

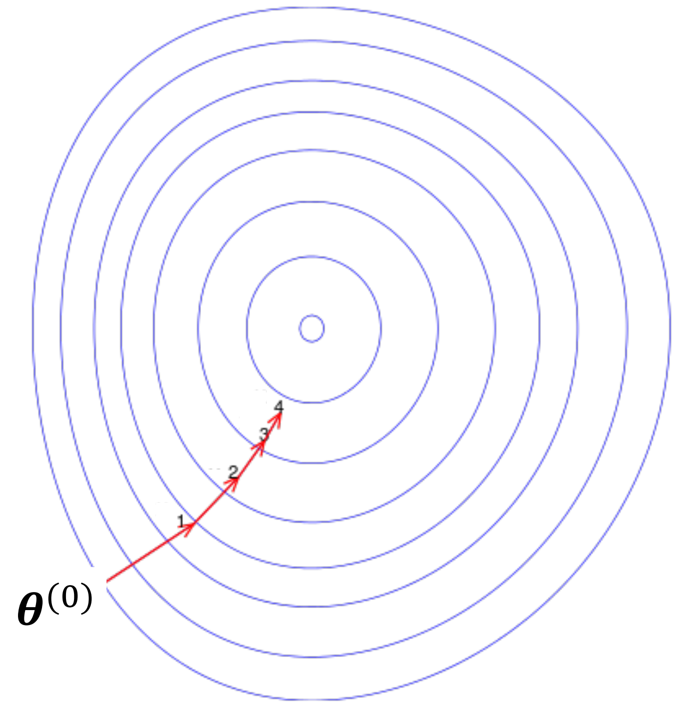
Harps, p. 984.

Gradient descent and SGD

1. Choose $\boldsymbol{\theta}^{(0)}$ and some T
2. For i from 1 to T*
   1. $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \eta \nabla L(\boldsymbol{\theta}^{(i-1)})$
3. Return $\widehat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^{(T)}$

Assuming L is differentiable

- Note: η dynamically updated per step
- Variants: Momentum, AdaGrad, …
- Stochastic gradient descent: two loops
  - Outer for loop: each loop (called epoch) sweeps through all training data
  - Within each epoch, randomly shuffle training data; then for loop: do gradient steps only on batches of data. Batch size might be 1 or few

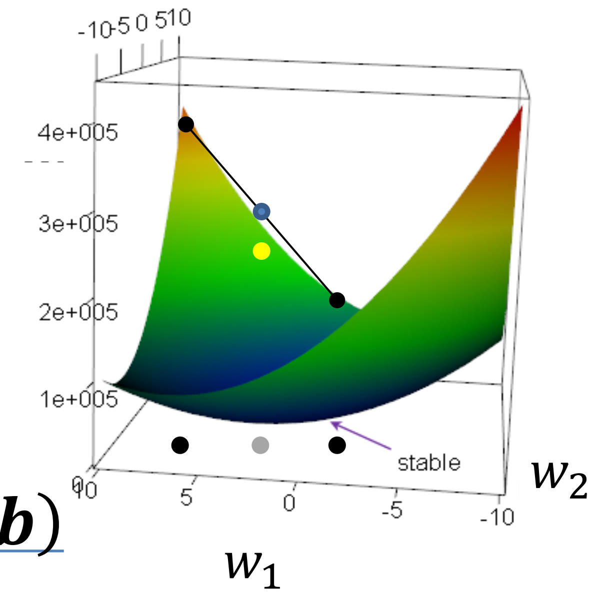


Wikimedia Commons. Authors: Olegalexandrov, Zerodamage

*Other stopping criteria can be used

Convex objective functions

- 'Bowl shaped' functions
- Informally: if line segment between any two points on graph of function lies above or on graph
- Formally* $f: D \to \mathbf{R}$ is convex if $\forall \boldsymbol{a}, \boldsymbol{b} \in D, t \in [0,1]$:
  $f(t\boldsymbol{a} + (1-t)\boldsymbol{b}) \leq tf(\boldsymbol{a}) + (1-t)f(\boldsymbol{b})$
  Strictly convex if inequality is strict (<)
- Gradient descent on (strictly) convex function guaranteed to find a (unique) global minimum!

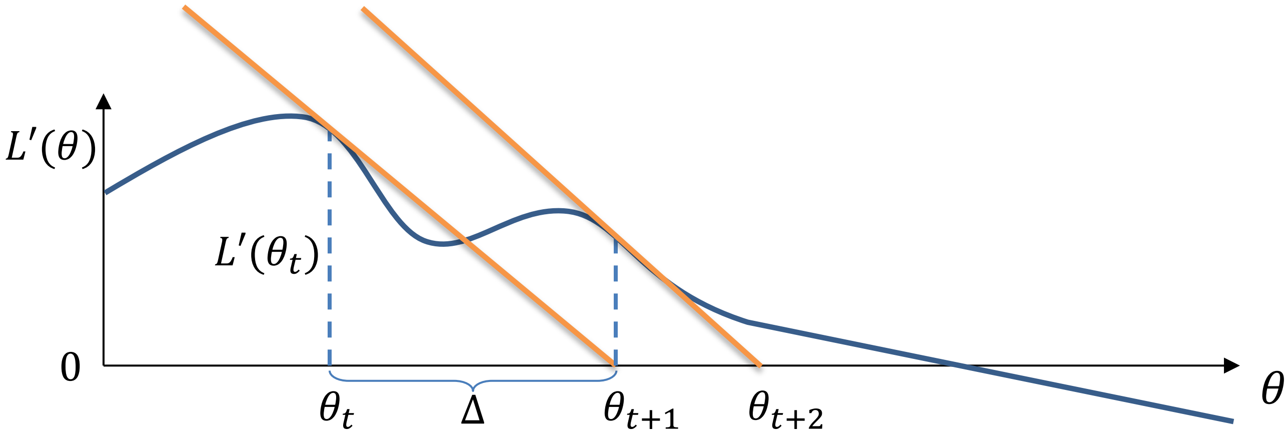


* Aside: Equivalently we can look to the second derivative. For *f* defined on scalars, it should be non-negative; for multivariate f, the Hessian matrix should be positive semi-definite (see linear algebra supplemental deck).

Newton-Raphson

A second-order method;
Successive root finding in the
objective’s derivative.

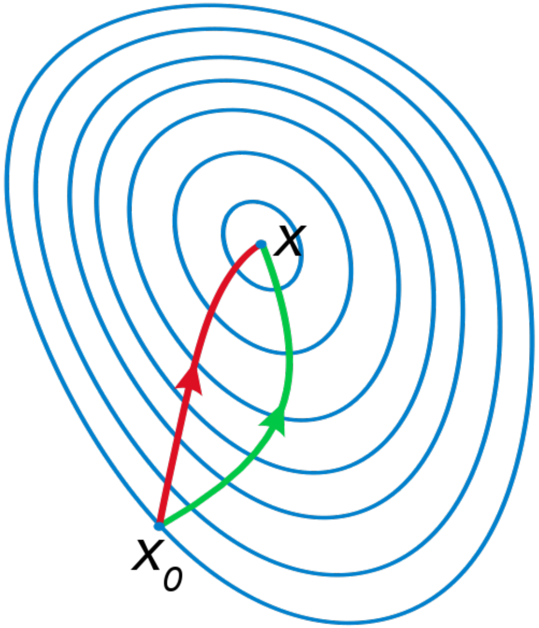
Newton-Raphson: Derivation (1D)



- Critical points of $L(\theta)$ = Zero-crossings of $L'(\theta)$
- Consider case of scalar θ. Starting at given/random θ_0, iteratively:
  1. Fit tangent line to $L'(\theta)$ at θ_t
  2. Need to find $\theta_{t+1} = \theta_t + \Delta$ using linear approximation's zero crossing
  3. Tangent line given by derivative: rise/run = $-L''(\theta_t) = L'(\theta_t)/\Delta$
  4. Therefore iterate is $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$

Newton-Raphson: General case

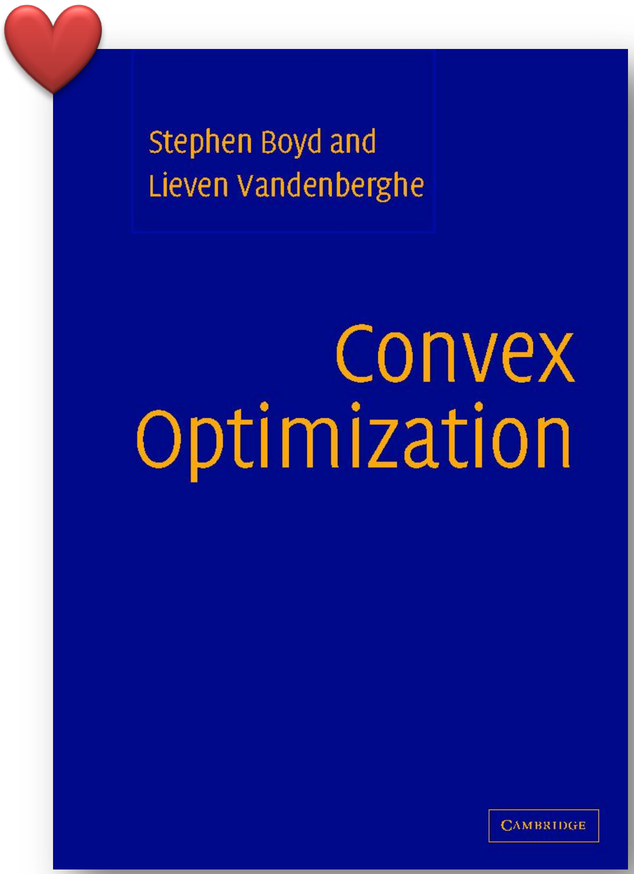
- Newton-Raphson summary
  - Finds $L'(\theta)$ zero-crossings
  - By successive linear approximations to $L'(\theta)$
  - Linear approximations involve derivative of $L'(\theta)$, ie. $L''(\theta)$
- Vector-valued θ:
  How to fix scalar $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$???
  - $L'(\theta)$ is $\nabla L(\theta)$
  - $L''(\theta)$ is $\nabla_2 L(\theta)$
  - Matrix division is matrix inversion
- General case: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \left(\nabla_2 L(\boldsymbol{\theta}_t)\right)^{-1} \nabla L(\boldsymbol{\theta}_t)$
  - Pro: May converge faster; fitting a quadratic with curvature information
  - Con: Sometimes computationally expensive, unless approximating Hessian



public domain wikipedia

…And much much more

- What if you have constraints?
  - See Lagrangian multipliers (let's you bring constraints into objective)
  - Or, projected gradient descent (you iterate between GD on objective, and GD on each constraints)
- What about speed of convergence?
- Do you really need differentiable objectives? (no, subgradients)
- Are there more tricks? (Hell yeah! But outside scope here)



Free at http://web.stanford.edu/~boyd/cvxbook/

Mini Summary

- Iterative optimisation for ML
  - First-order: Gradient Descent and Stochastic GD
  - Convex objectives: Convergence to global optima
  - Second-order: Newton-Raphson can be faster, can be expensive to build/invert full Hessian

Next: Logistic regression for binary classification

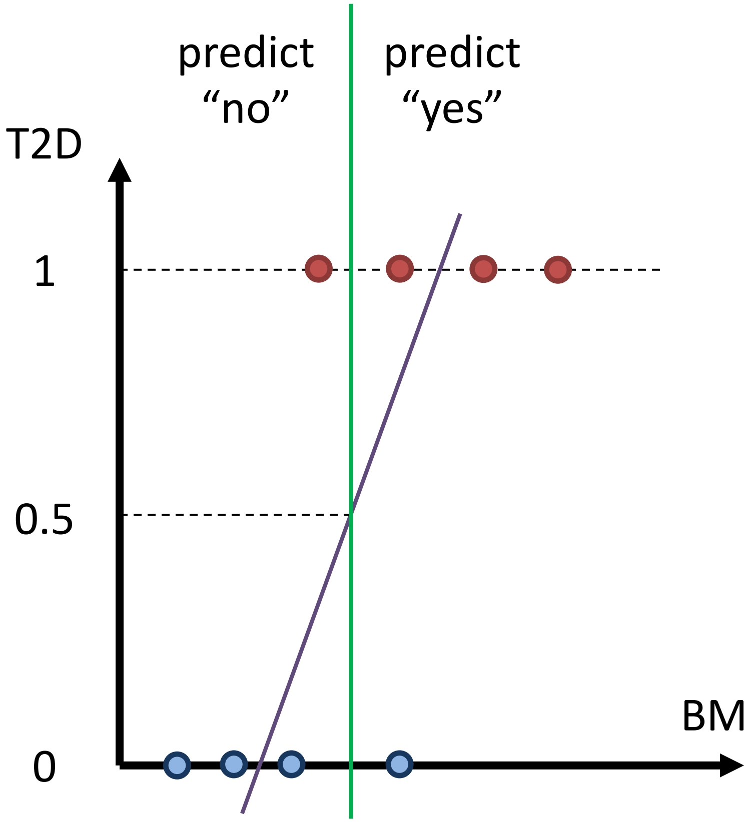
Logistic Regression Model

A workhorse linear, binary classifier;

(A review for some of you; new to some.)

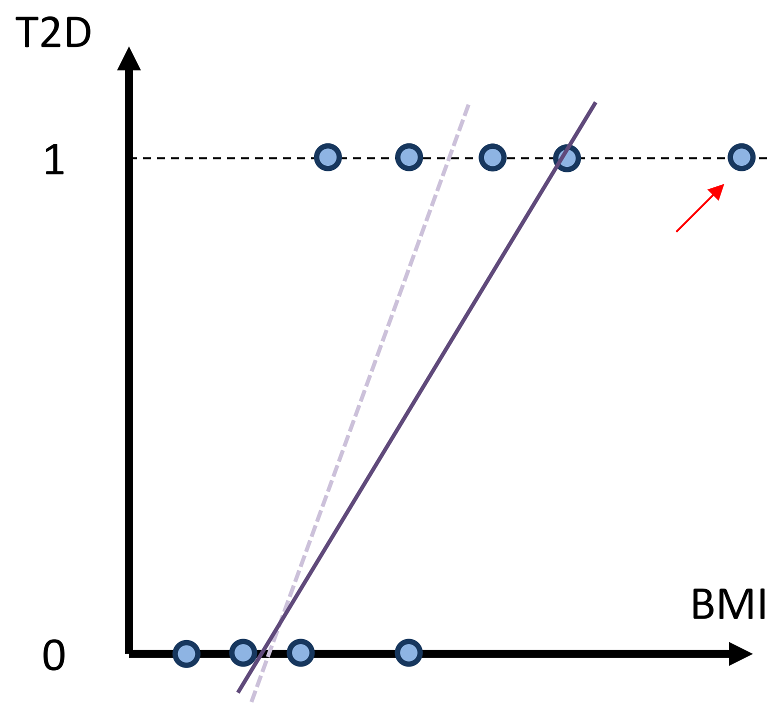
Binary classification: Example

- Example: given body mass index (BMI) does a patient have type 2 diabetes (T2D)?
- This is (supervised) binary classification
- One *could* use linear regression
  - Fit a line/hyperplane to data (find weights $\boldsymbol{w}$)
  - Denote $s \equiv \boldsymbol{x}'\boldsymbol{w}$
  - Predict "Yes" if $s \geq 0.5$
  - Predict "No" if $s < 0.5$



Why not linear regression

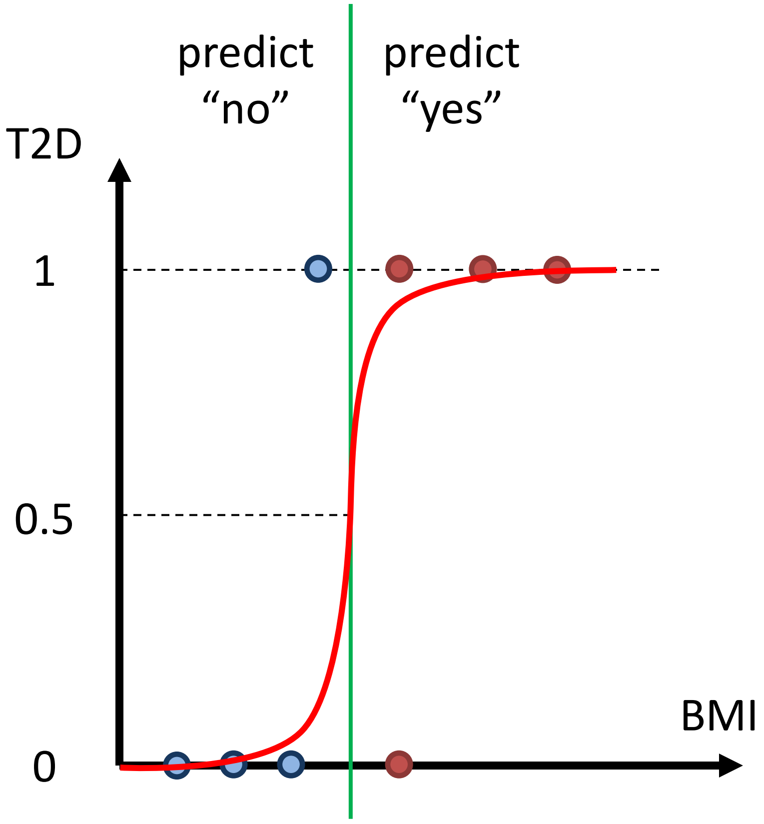
- Due to the square loss, points far from boundary have loss squared – even if they're confidently correct!
- Such "outliers" will "pull at" the linear regression
- Overall, the least-squares criterion looks unnatural in this setting



Logistic regression model

- Probabilistic approach to classification
  - $P(Y=1|\boldsymbol{x}) = f(\boldsymbol{x}) = ?$
  - Use a linear function? E.g., $s(\boldsymbol{x}) = \boldsymbol{x}'\boldsymbol{w}$
- Problem: the probability needs to be between 0 and 1.
- Logistic function $f(s) = \frac{1}{1+\exp(-s)}$
- Logistic regression model

$$P(Y=1|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{x}'\boldsymbol{w})}$$



How is logistic regression *linear*?

- Logistic regression model:

$$P(Y=1|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{x}'\boldsymbol{w})}$$

- Classification rule:

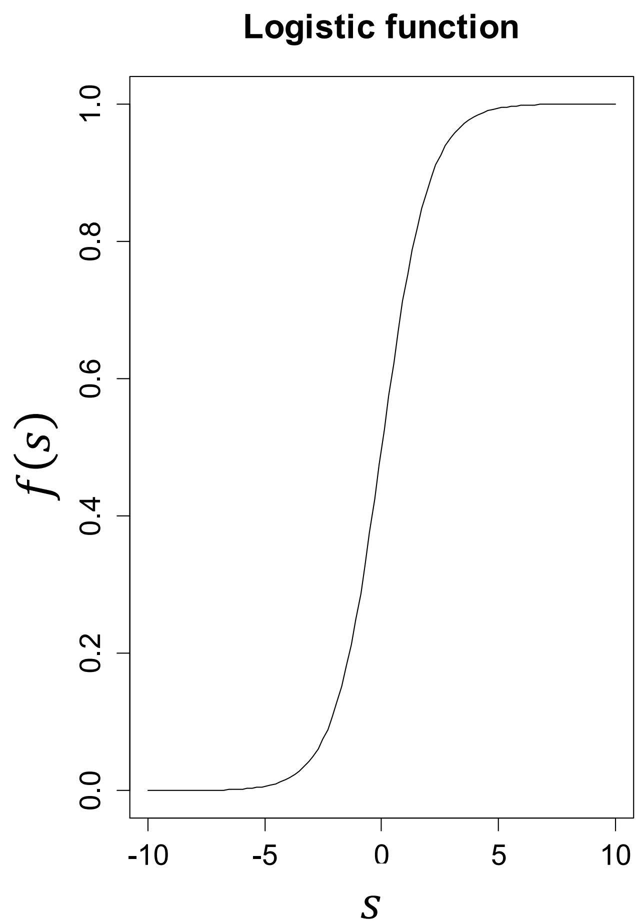
if $\left(P(Y=1|\boldsymbol{x}) > \frac{1}{2}\right)$ then class "1", else class "0"

- Decision boundary is the set of **x**'s such that:

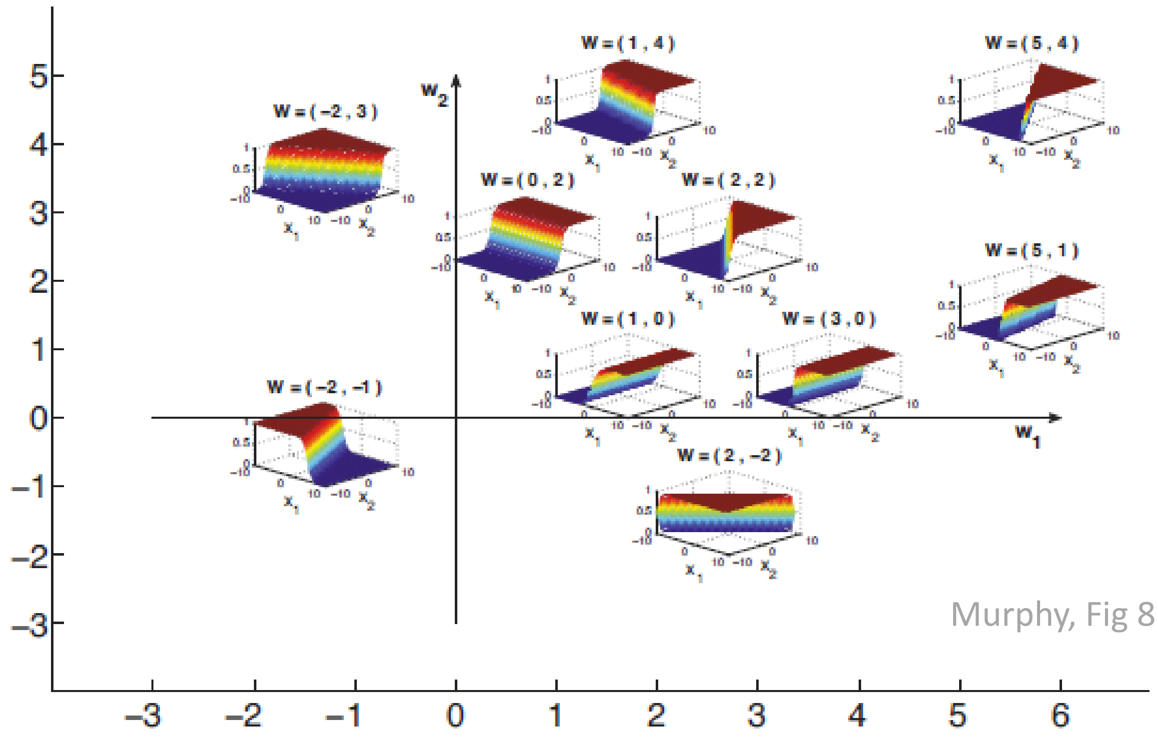
$$\frac{1}{1+\exp(-\boldsymbol{x}'\boldsymbol{w})} = \frac{1}{2}$$

$$\exp(-w'x) = 1$$

$$w'x = 0$$



Effect of parameter vector (2D problem)



Murphy, Fig 8.1, p246

- Decision boundary is the line where $P(Y=1|\boldsymbol{x})=0.5$
  - In higher dimensional problems, the decision boundary is a plane or hyperplane
- Vector $\boldsymbol{w}$ is perpendicular to the decision boundary (see linear algebra review topic)
  - That is, $\boldsymbol{w}$ is a normal to the decision boundary
  - Note: in this illustration we assume $w_0=0$ for simplicity

Linear vs. logistic probabilistic models

- Linear regression assumes a Normal distribution with a fixed variance and mean given by linear model

$$p(y|\boldsymbol{x}) = Normal(\boldsymbol{x}'\boldsymbol{w}, \sigma^2)$$

- Logistic regression assumes a Bernoulli distribution with parameter given by logistic transform of linear model

$$p(y|\boldsymbol{x}) = Bernoulli(\text{logistic}(\boldsymbol{x}'\boldsymbol{w}))$$

- Recall that Bernoulli distribution is defined as

$$p(1) = \theta \text{ and } p(0) = 1 - \theta \text{ for } \theta \in [0,1]$$

- Equivalently $p(y) = \theta^y(1-\theta)^{(1-y)}$ for $y \in \{0,1\}$

Training as Max-Likelihood Estimation

- Assuming independence, probability of data

$$p(y_1, \dots, y_n | \boldsymbol{x}_1, \dots, \boldsymbol{x}_n) = \prod_{i=1}^{n} p(y_i | \boldsymbol{x}_{\boldsymbol{i}})$$

- Assuming Bernoulli distribution we have

$$p(y_i | \boldsymbol{x}_i) = \left(\theta(\boldsymbol{x}_i)\right)^{y_i} \left(1 - \theta(\boldsymbol{x}_i)\right)^{1-y_i}$$

$$\text{where } \theta(\boldsymbol{x}_i) = \frac{1}{1+\exp(-\boldsymbol{x}_i{}'\boldsymbol{w})}$$

- Training: maximise this expression wrt weights $\boldsymbol{w}$

Apply log trick, simplify

- Instead of maximising likelihood, maximise its logarithm

$$\log\left(\prod_{i=1}^{n} p(y_i|\boldsymbol{x_i})\right) = \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x_i})$$

$$= \sum_{i=1}^{n} \log\left(\left(\theta(\boldsymbol{x}_i)\right)^{y_i}\left(1-\theta(\boldsymbol{x}_i)\right)^{1-y_i}\right)$$

$$= \sum_{i=1}^{n}\left(y_i \log\left(\theta(\boldsymbol{x}_i)\right) + (1-y_i)\log\left(1-\theta(\boldsymbol{x}_i)\right)\right)$$

$$= \sum_{i=1}^{n}\left((y_i - 1)\boldsymbol{x}_i'\boldsymbol{w} - \log(1+\exp(-\boldsymbol{x}_i'\boldsymbol{w}))\right)$$

Can't do this analytically

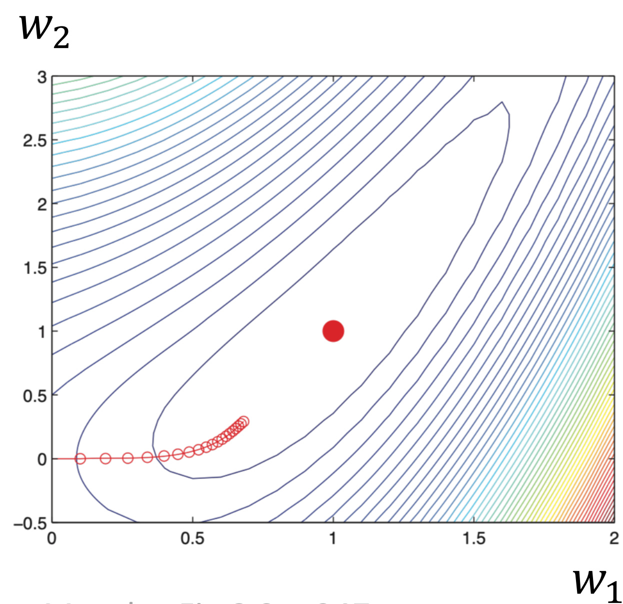
Training as Iterative Optimisation

- Training logistic regression: $\mathbf{w}$ maximising log-likelihood $L(\mathbf{w})$ or cross-entropy loss
- **Bad news**: No closed form solution
- **Good news**: Problem is strictly convex, if no irrelevant features → convergence!

Look ahead: regularisation for irrelevant features

How does gradient descent work?

- simply take gradient of log-likelihood, i.e., $\nabla L(\mathbf{w}) = \sum_{i=1}^{n}\left(y_i - \theta(\mathbf{x}_i)\right)\mathbf{x}_i$
- plug into favourite iterative optimiser GD/SGD/Adagrad/Adam/BFGS/…



Murphy, Fig 8.3, p247

Logistic Regression: Decision-Theoretic View

Via cross-entropy loss

Background: Cross entropy

- Cross entropy is an information-theoretic method for comparing two distributions
- Cross entropy is a measure of a divergence between reference distribution $g_{ref}(a)$ and estimated distribution $g_{est}(a)$. For discrete distributions:

$$H\left(g_{ref}, g_{est}\right) = -\sum_{a \in A} g_{ref}(a) \log g_{est}(a)$$

A is support of the distributions, e.g., $A = \{0,1\}$

Training as cross-entropy minimisation

- Consider log-likelihood <u>for a single data point</u>

$$\log p(y_i|\boldsymbol{x_i}) = y_i \log\left(\theta(\boldsymbol{x}_i)\right) + (1 - y_i)\log\left(1 - \theta(\boldsymbol{x}_i)\right)$$

- Cross entropy $H\left(g_{ref}, g_{est}\right) = -\sum_a g_{ref}(a) \log g_{est}(a)$
  - If reference (true) distribution is

$$g_{ref}(1) = y_i \text{ and } g_{ref}(0) = 1 - y_i$$

  - With logistic regression estimating this distribution as

$$g_{est}(1) = \theta(\boldsymbol{x}_i) \text{ and } g_{est}(0) = 1 - \theta(\boldsymbol{x}_i)$$

It finds $\boldsymbol{w}$ that minimises sum of cross entropies per training point

Mini Summary

- Logistic regression *formulation*
  - A workhorse linear binary classifier (but can use basis function to become non-linear)
  - Frequentist: Bernoulli label with coin bias logistic-linear in $\mathbf{x}$
  - Decision theory: Minimising cross entropy with labels

Next time: Regularised linear regression for avoiding overfitting and ill-posed optimisation