For the timing being, our goal is binary classification.

A *training set* $\mathcal{T}$ is given with known classification.

- Step 1: Using a probabilistic model, write a function out of $\mathcal{T}$ with unknown *parameters* or *weights*.
- Step 2: Determine the parameters so that the known classification may have the maximum likelihood.

$$\text{Learning} \iff \text{Maximizing Certainty}$$

Finding a maximum or a minimum is one of the main topics in Machine Learning. It is called optimization.

It is customary to take the negative log of the likelihood function.

Later we will see that the resulting function is given by

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\},$$

$$y_n = \sigma(w_1 x_{n1} + w_2 x_{n2} + \cdots + w_k x_{nk} + w_{k+1}).$$

We need to *minimize* this function.

How??

We use gradient descent or Newton's method.

These methods can be applied to many other functions.

Newton's Method

- Second-order approximation

  Much faster in convergence, more expensive (and more subtle)

- $f(x)$: single-variable (convex, differentiable) function

  Find a local minimum

  $$\iff \qquad \text{Find } x_* \text{ such that } f'(x_*) = 0$$

  Make a guess x_0 for x_* and set $x = x_0 + h$.

- Using Taylor's expansion,

$$\begin{aligned} f(x) &= f(x_0+h) \approx f(x_0) + f'(x_0)h + \tfrac{1}{2}f''(x_0)h^2 \\ f'(x) &\approx \frac{d}{dh}\left(f(x_0) + f'(x_0)h + \tfrac{1}{2}f''(x_0)h^2\right) \\ &= f'(x_0) + f''(x_0)h \end{aligned}$$

From $f'(x) = 0$, we approximately obtain

$$0 = f'(x_0) + f''(x_0)h, \qquad h = -f'(x_0)/f''(x_0).$$

- We have shown that

$$x_1 = x_0 - f'(x_0)/f''(x_0)$$

  is an approximation of x^*.
- Repeat the process to obtain

$$\boxed{x_{k+1} = x_k - f'(x_k)/f''(x_k)},$$

  and $x_k \to x_*$ as $k \to \infty$.
- This is Newton's method for a single-variable function, and we generalize it to a multi-variable function.

- $F(\boldsymbol{x})$: multi-variable function (convex, differentiable)

  The Hessian matrix is defined by

$$\mathbf{H}F = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_m} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_m \partial x_1} & \frac{\partial^2 F}{\partial x_m \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_m^2} \end{bmatrix}.$$

  That is, $\mathbf{H}F = [\frac{\partial^2 F}{\partial x_i \partial x_j}]$.

- Recall
$$x_{k+1} = x_k - f'(x_k)/f''(x_k).$$
- Generalizing the single-variable case,
$$\boxed{\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mathbf{H}F(\boldsymbol{x}_k)^{-1}\nabla F(\boldsymbol{x}_k)}.$$

Proof: Using Taylor's expansion, we have for $\boldsymbol{h} \in \mathbb{R}^m$,

$$\begin{aligned} F(\boldsymbol{x}) &= \boxed{F(\boldsymbol{x}_0 + \boldsymbol{h}) \approx F(\boldsymbol{x}_0) + \boldsymbol{h}^\top \nabla F(\boldsymbol{x}_0) + \tfrac{1}{2}\boldsymbol{h}^\top \mathbf{H}F(\boldsymbol{x}_0)\boldsymbol{h}} \\ \frac{\partial F}{\partial x_i}(\boldsymbol{x}) &\approx \frac{\partial}{\partial h_i}\left(F(\boldsymbol{x}_0) + \boldsymbol{h}^\top \nabla F(\boldsymbol{x}_0) + \tfrac{1}{2}\boldsymbol{h}^\top \mathbf{H}F(\boldsymbol{x}_0)\boldsymbol{h}\right) \\ &= \frac{\partial F}{\partial x_i} + \sum_{k=1}^{n} H_{ik} h_k, \end{aligned}$$

where we write $\mathbf{H}F(\boldsymbol{x}_0) = [H_{k\ell}]$. Thus

$$\nabla F(\boldsymbol{x}) \approx \nabla F(\boldsymbol{x}_0) + \mathbf{H}F(\boldsymbol{x}_0)\boldsymbol{h}.$$

From $\nabla F(\boldsymbol{x}) = 0$, we approximately obtain

$$\boldsymbol{h} = -\mathbf{H}F(\boldsymbol{x}_0)^{-1}\nabla F(\boldsymbol{x}_0).$$

□

- Using a step size η_k, the formula may be modified to be

$$\boxed{\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \mathbf{H}F(\boldsymbol{x}_k)^{-1}\nabla F(\boldsymbol{x}_k)}.$$

- Newton's method is much faster than gradient descent. However, it may be expensive to compute $\mathbf{H}F(\boldsymbol{x}_k)^{-1}$. Sometimes, $\mathbf{H}F(\boldsymbol{x}_k)$ is close to a singular matrix.

Example

- Consider $E(\boldsymbol{w}) = E(w_1, w_2) = w_1^4 + w_2^4 - 16w_1w_2$.

  Then $\nabla E(\boldsymbol{w}) = [4w_1^3 - 16w_2, 4w_2^3 - 16w_1]^\top$.

$$\mathbf{H}E(\boldsymbol{w}) = \begin{bmatrix} 12w_1^2 & -16 \\ -16 & 12w_2^2 \end{bmatrix}$$

$$\mathbf{H}E(\boldsymbol{w})^{-1} = \frac{1}{9w_1^2w_2^2 - 16} \begin{bmatrix} \frac{3}{4}w_2^2 & 1 \\ 1 & \frac{3}{4}w_1^2 \end{bmatrix}$$

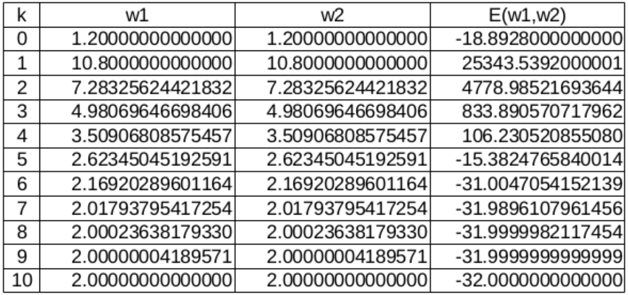
$$\mathbf{H}E^{-1}\nabla E = \frac{1}{9w_1^2w_2^2 - 16} \begin{bmatrix} 3w_1^3w_2^2 - 8w_2^3 - 16w_1 \\ 3w_1^2w_2^3 - 8w_1^3 - 16w_2 \end{bmatrix}$$

- Choose $\boldsymbol{w}_0 = (1, 1)$ and $\eta = 1$.

  Then $\boldsymbol{w}_1 = (2, 2)$ and $E(\boldsymbol{w}_1) = -32$.
- Choose $\boldsymbol{w}_0 = (1.2, 1.2)$ and $\eta = 1$.

  Then $\boldsymbol{w}_9 = (2.00000004189571, 2.00000004189571)$,

  $E(\boldsymbol{w}_9) = -31.9999999999999$.



k	w1	w2	E(w1,w2)
0	1.20000000000000	1.20000000000000	-18.8928000000000
1	10.8000000000000	10.8000000000000	25343.5392000001
2	7.28325624421832	7.28325624421832	4778.98521693644
3	4.98069646698406	4.98069646698406	833.890570717962
4	3.50906808575457	3.50906808575457	106.230520855080
5	2.62345045192591	2.62345045192591	-15.3824765840014
6	2.16920289601164	2.16920289601164	-31.0047054152139
7	2.01793795417254	2.01793795417254	-31.9896107961456
8	2.00023638179330	2.00023638179330	-31.9999982117454
9	2.00000004189571	2.00000004189571	-31.9999999999999
10	2.00000000000000	2.00000000000000	-32.0000000000000

- Apply Newton's method to our main example:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\},$$

  where $y_n = \sigma(w_1 x_{n1} + w_2 x_{n2} + \cdots + w_k x_{nk} + w_{k+1})$.
- Recall that $\sigma'(x) = \sigma(x)(1-\sigma(x))$.

- We have

$$\nabla E(\boldsymbol{w}) = \left[\sum_{n=1}^{N}(y_n - t_n)x_{nj}\right] = X^\top(\boldsymbol{y} - \mathbf{t}),$$

where

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1k} & 1 \\ x_{21} & \cdots & x_{2k} & 1 \\ \vdots & & \vdots & \vdots \\ x_{N1} & \cdots & x_{Nk} & 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \text{and } \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}.$$

- $$\frac{\partial^2 E}{\partial w_i \partial w_j} = \sum_{n=1}^{N} y_n(1 - y_n)x_{ni}x_{nj}$$
- We get
$$\mathbf{H}E = \left[\sum_{n=1}^{N} y_n(1 - y_n)x_{ni}x_{nj}\right] = X^\top R X,$$
where $R = \mathrm{diag}(y_n(1 - y_n))$.
- Then we have
$$\boxed{\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - (X^\top R X)^{-1} X^\top (\boldsymbol{y} - \mathbf{t})},$$
where R and $\boldsymbol{y}$ are determined by $\boldsymbol{w}_k$ in each step.

Stochastic Gradient Descent (SGD)

- Typically in Machine Learning, we minimize a function $E(\boldsymbol{w})$ given by a sum of the form

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} E_n(\boldsymbol{w}),$$

where N is the number of elements in the training set.

- When N is large, computation of the gradient ∇E may be expensive.

- The SGD selects a sample from the training set in each iteration step instead of using the whole batch of the training set, and use

$$\frac{1}{M}\sum_{i=1}^{M}\nabla E_{n_i}(\boldsymbol{w}),$$

  where M is the size of the sample and $\{n_1, n_2, \ldots, n_M\} \subset \{1, 2, \ldots, N\}$.
- The SGD is commonly used in many Machine Learning algorithms.