Week3 - Linear Regression Models

- Polynomial is a specific example of a broad class of function call Linear Regression Models
- Having the property of being linear functions of the adjustable parameters
- The simplest form of the linear regression models which is also linear to the input variable: $y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x + \cdots + w_{M-1} x$ (the simplest basis function is $\phi_i(x) = x$.)

1. Basis Function

- Usually taking linear combination of a fixed set of **nonlinear functions of the input variables**, known as **basis functions**.
- **Aim**: predict the value of continuous target variable $\boldsymbol{t}$
- **Input**: d-dimensional vector $\boldsymbol{x}$ of input variables
- Generalised form:
  $y(x, \boldsymbol{w}) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \cdots + w_{M-1} \phi_{M-1}(x) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) = \sum_{j=0}^{M-1} w_j \phi_j(x)$
  - If $\phi_i(x)$ is linear of x, it is called **linear basis function**
  - Otherwise, it is called **non-linear basis function**
- Basis function provide a way of making a linear regression model nonlinear
- Dummy basis function: $\phi_0(x) = 1$

Popular Basis functions

- Gaussian: $\phi_j(x) := \exp\left(-\frac{(x - \mu_j)^2}{2s^2}\right)$
- Sigmoidal: $\phi_j(x) := \sigma\left(\frac{x - \mu_j}{s}\right)$
  - Logistic sigmoid function: $\sigma(a) := \frac{1}{1 + \exp(-a)}$
  - Hyperbolic tangent function: $\tanh(a) = 2\sigma(2a) - 1 = \frac{1 - e^{-2a}}{1 + e^{-2a}}$

2. Loss function

We fit a regression model to the training data by minimising an error function that measures the misfit between the predictions $y(\boldsymbol{x}, \boldsymbol{w})$ for each data training point x_n and the target value t_n.

- Calculate the misfit: SSE - the sum of the squares of the errors

  $E(\boldsymbol{w}) := \frac{1}{2} \sum_{n=1}^{N} [y(x_n, \boldsymbol{w}) - t_n]^2$

  It shifts from minimising the error function to finding $\boldsymbol{w}$ that satisfies the function:

  $\boldsymbol{w} := \arg\min_{\boldsymbol{w}} E(\boldsymbol{w})$
- Components:
  - **Noise**: Data collected by sampling from ensemble have inevitably included noise:

$\epsilon = N(0, \sigma^2)$. Later we will find It represents the irreducible minimum value of the loss function.

- Model Function: $y(\boldsymbol{x}, \boldsymbol{w})$
- Observed target value: $\boldsymbol{t} := \{t_n\}$ where $n \in [1, N]$
- Assume the training data points are drawn independently (**i.i.d**) from the underlying ensemble: $\boldsymbol{x} := \{x_n\}$,where $n \in [1, N]$

Then, The likelihood:

$p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \sigma^2) = \prod_N^{n=1} N(t_n|y(x_n, \boldsymbol{w}), \sigma^2)$

The log-likelihood function:

$\ell(\boldsymbol{w}) = \log\left(p(\boldsymbol{t}|\boldsymbol{x}, \boldsymbol{w}, \sigma^2)\right) = \sum_N^{n=1} \log N(t_n|y(x_n, \boldsymbol{w}), \sigma^2)$

Let use precision to replace variance: $\beta^{-1} = \sigma^2$

$$\ell(\boldsymbol{w}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi - \beta E(\boldsymbol{w})$$

Note that only the last term relevant to the function $\boldsymbol{w}$.

Then differentiates the equation with respect to $\boldsymbol{w}$, we can have the gradient of the log likelihood:

$\nabla\ell(\boldsymbol{w}) = \beta\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\phi(x_n)\}\phi(x_n)^T$

Let the gadient equals to zero, we can find the best $\boldsymbol{w}$ that maximise the log-likelihood function:

$$\nabla\ell(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial\ell(\boldsymbol{w})}{\partial w_0} \\ \frac{\partial\ell(\boldsymbol{w})}{\partial w_1} \\ \cdots \\ \frac{\partial\ell(\boldsymbol{w})}{\partial w_{M-1}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \Rightarrow \boldsymbol{w} = (\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$$

> We can find the parameter by solving the matrix, elegant but expensive

3. Iterative Optimization Algorithms

- Step by step:
  - Initialise the parameters at random
  - Iteratively moves towards the negative direction of the objective function gradient to find the optimal parameters

    > Gradient: 梯度是一个矢量，其的方向取决于是其方向导数中最大值的方向，梯度的值是方向导数的最大值（最陡）。
    >
    > 需要注意的是，梯度指的是增长最快的方向。因此对于梯度下降，我们需要朝某点梯度的反方向前进才能得到其驻点。

- **Gradient Descent Algorithms (with Batch technique: BGD)**:
  - Pseudocode:

```
# Gradient Descent
Initialise w and t = 1, t_max = N
Initialise eta and epslion #learning rate & theshold
While (diff > epslion and t <= t_max)
do{
    w_new := w - eta * gradient(w,X)
    diff := gradient(w_new, X) - gradient(w,X)
    if diff < 0 then break
    else eta = eta * 0.5 #control learning rate
    w := w_new # update
    t = t + 1 #increment
}
```

- Error function: $E(\boldsymbol{w}) = \sum_{n=1}^{n} E_n(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[t_n - \boldsymbol{w}\cdot\phi(x_n)]^2$
- Gradient: $\nabla E(\boldsymbol{w}) = -\sum_{n=1}^{N}[t_n - \boldsymbol{w}\cdot\phi(x_n)]\cdot\phi(x_n)$
- An algorithm that minimises functions. It starts with an initial set of parameter value $\boldsymbol{w}^{(0)}$, and iteratively moves towards optimal parameters which achieve the minimum value of the objective function.
- Process the **entire training set** at each iteration.
- Computationally **costly** for large data sets
- Not guarantee to converge: sensitive to starting points, may reach local minima instead of global minima

- **Stochastic Gradient Descent Algorithms (SGD)**:
  - Pseudocode

```
# Stochastic Gradient Descent
Initialise w and t = 1, t_max = N
Initialise eta and epslion #learning rate & theshold
While (diff > epslion and t <= t_max)
do{
    for each training point x_i
    w_new := w - eta * gradient(w,X)
    diff := gradient(w_new, X) - gradient(w,X)
    if diff < 0 then break
    else eta = eta * 0.5 # control learning rate
    w := w_new # update
    t = t + 1 #increment
}
```

  - Error function: $E(\boldsymbol{w}) = \sum_{n=1}^{n} E_n(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[t_n - \boldsymbol{w}\cdot\phi(x_n)]^2$
  - Gradient: $\nabla E_n(\boldsymbol{w}) = -[t_n - \boldsymbol{w}^T\cdot\boldsymbol{\phi(x_n)}]\cdot\boldsymbol{\phi(x_n)}$
  - Process **one data points** at each iteration $E(\boldsymbol{w}) = \sum_{n=1}^{n} E_n(\boldsymbol{w})$
  - **Incrementally update** model parameters after each such presentation
    - known as **least-mean-squares (LMS)**: $\boldsymbol{w}^t = \boldsymbol{w}^{t-1} - \eta\cdot\nabla E_n(\boldsymbol{w}^{t-1})$
    - The value of η needs to be chosen with care to **ensure that the algorithm**

converges.
    - Not necessary to process the entire data set
    - **Faster than BGD** to get $\boldsymbol{w}$ **close to the minimum** but not **never converge** to the minimum (keep **oscillating around** the minimum of the objective function)
    - Sensitive to **feature scaling**, need to standardise training data
- **Problem**:
    - Need to compute the gradient of the error function $\nabla E(\boldsymbol{w})$.
    - Need to consider the **starting point** $\boldsymbol{w}^{(0)}$, the **learning rate** η , the **threshold** ϵ . the **maximum iteration number** t_{max}
        - Statring point $\boldsymbol{w}^{(0)}$: reach local minima
        - Learning rate (step-size) η :
            - Too large:
            - Too small: too slow to convergence
        - Threshold ϵ: criteria to stop the update process
        - Maximum iteration number t_{max}: criteria to stop the update process

4. Regularisation

- The total error function with regularization term to control over-fitting: $E_D(\boldsymbol{w}) + \lambda E_W(\boldsymbol{w})$
- The general regularization term: $E_W(\boldsymbol{w}) = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|^q$
    - The simplest regularization term given by the sum-of-squares of the weight vector elements(paramters): $E_W(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|^2$
    - Different q gives different regularization function. **The choice of** q **depend on problems**.

L2 Ridge Regularisation: $q = 2$

- Known as **weight decay**: encourage weight values to **decay towards zero**, unless supported by the data.
- Advantage: the error function remains a quadratic function of $\boldsymbol{w}$, the exact minimizer can be found in closed form: $\boldsymbol{w} = \left(\lambda\boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$

L1 Lasso Regularisation: $q = 1$

- $E_W(\boldsymbol{w}) = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|$
- If λ is sufficiently large, some of the coefficient w_j are driven to zero, leading to a **sparse model in which the corresponding basis functions play no role**
- As λ is increased, an increasing number of parameters are driven to zero.
- Can be use for **feature selection**

Key Difference: L2 v.s L1

- **L2 Ridge**: Mainly for **prevent overfitting**, **not useful for large number of input variables**.
    - includes all of the input variables in the model
    - Parameter shrinkage (limit the magnitude)
    - Reduce model complexity
- **L1 Lasso**: Mainly for **sparse solution**, **useful for large number of input variables**.
    - Parameter shrinkage
    - Some coefficients are driven to zero (**feature selection**: some input variables are excluded)

> Note: Limiting the number of basis functions in order to avoid over-fitting has the side effect of limiting the flexibility of the model to capture interesting and important trends in the data.