機器學習 Machine Learning

劉昆琳 111061528 HW3 05/22/2023

Part 2. Programming assignment :

1. Maximum Likelihood Estimation

假設資料的分布如圖 1 所示，則期望找到迴歸線的參數有斜率(W)及偏差(b)，此迴歸線根據課本將其表示成$y(\boldsymbol{x},\boldsymbol{w})$，其中$\boldsymbol{x}$代表著特徵向量，而$\boldsymbol{w}$則代表擬合的多項式參數，但擬合出來的曲線並不會完美的與資料的分布重疊，故在假設模型的函數時，會加上一個平均值為 0、標準差為β^{-1}的高斯雜訊ϵ，即$t = y(\boldsymbol{x},\boldsymbol{w}) + \epsilon$，且雜訊的 probability distribution 則可以表達成$p(t|\boldsymbol{x},\boldsymbol{w},\beta) = \mathcal{N}(t|y(\boldsymbol{x},\boldsymbol{w}),\beta^{-1})$。

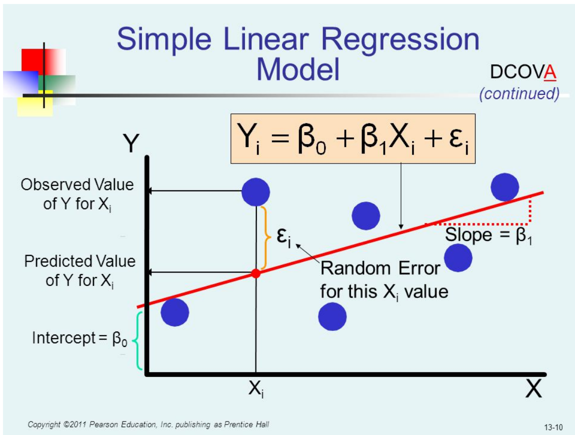

圖 1、資料集的分布與模型建模

因為樣本之間相互獨立，所以聯合機率(joint probability)可以被表達成$p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w},\beta) = \prod_{n=1}^{N}\mathcal{N}(t_n|w^T\phi(x_n),\beta^{-1})$，其中$y(\boldsymbol{x},\boldsymbol{w})$被改寫成矩陣的形式，$w$為多項式參數，$\phi$則是多項式，如$1$、$x$、$x^2$、$x^3\cdots$，由於高斯函數中有指數不好推導，故將其取 ln，即

$$ln\big(p(\boldsymbol{t}|\boldsymbol{w},\beta)\big) = \sum_{n=1}^{N} ln\big(\mathcal{N}(t_n|w^T\phi(x_n),\beta^{-1})\big) = \frac{N}{2}ln(\beta) - \frac{N}{2}ln(2\pi) - \beta E_D(\boldsymbol{w})$$

其中$E_D(\boldsymbol{w})$為

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - w^T\phi(x_n)\}^2$$

若想擬合出來的函數越接近理想，則目標為最小化高斯雜訊ϵ，也就是最小化$ln\big(p(\boldsymbol{t}|\boldsymbol{w},\beta)\big)$，故對其取$\boldsymbol{w}$的偏微分，並令導數等於零，如下:

$$\nabla\, ln\big(p(\boldsymbol{t}|\boldsymbol{w},\beta)\big) = \beta\sum_{n=1}^{N}\{t_n - w^T\phi(x_n)\}\phi(x_n)^T$$

$$0 = \sum_{n=1}^{N} t_n\phi(x_n)^T - w^T\sum_{n=1}^{N}\phi(x_n)\phi(x_n)^T$$

即可求得$\boldsymbol{w}$為$(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$，而$\boldsymbol{\Phi}$是根據課本 3.16 定義的 design matrix，其中每一列$x^{(i)}$代表的是第i筆樣本，每一行x_j代表的是第j個特徵；透過以上敘述，將 Maximum Likelihood Regression 寫成一個類別 MLR()，如圖 2 所示，而在測試集上計算 MSE 的結果如圖 3。

```python
class MLR:
    def __init__(self):

        self.weights = None

    def fit(self, x, y):

        design_matrix = self.compute_design_matrix(x)

        design_matrix_dot_product = design_matrix.T @ design_matrix

        y_reshaped = y.reshape(-1, 1)

        self.weights = np.linalg.inv(design_matrix_dot_product) @ design_matrix.T @ y_reshaped

    def predict(self, x):

        design_matrix = self.compute_design_matrix(x)

        pred_mean = design_matrix.dot(self.weights)

        return pred_mean

    def compute_design_matrix(self, features: np.ndarray) -> np.ndarray:
        """
        Args:
            features: numpy array of features
        Returns:
            design_matrix: numpy array of transformed features
        """
        n_samples = len(features)
        phi_0 = np.ones(n_samples).reshape(-1, 1)
        design_matrix = np.concatenate([phi_0, features], axis=1)
        return design_matrix
```

圖 2、Maximum Likelihood Regression

```
(MLR)Testing set's MSE : 126.07742667558452
```

圖 3、MLR 在測試集上的 MSE 結果

2. Bayesian Linear Regression

在 Bayesian 的觀點中，權重$\boldsymbol{w}$是一個隨機變量，通過考慮參數的 prior probability 和觀測數據的 likelihood，獲得參數的 posterior probability，從而進行參數估計，也就是要求出$\boldsymbol{p}(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{Y})$，推導如圖 4。

求 $p(w|X,Y)$。

∵ w 與訓練集 Data X 獨立 $= P(w)$

$$p(w|X,Y) = \frac{P(Y|w,X)\cdot P(w|X)}{P(Y|X)}$$

$$= \frac{P(Y|w,X)\cdot P(w)}{\int P(Y|w,X)\cdot P(w)\,dw}$$ 常量

∵ N 個數據皆相互獨立

$$\therefore p(Y|w,X) = \prod_{i=1}^{N} p(y_i|w_i x_i)$$

$$= \prod_{i=1}^{N} N(y_i|w^T x_i, \sigma^2)$$

∵ $p(w)$ 假設為一隨機變量 $\Rightarrow N(w|m_0, S_0)$

Gaussian 的共軛特質 ⇒ prior 是 Gaussian，likelihood 也是 Gaussian ⇒ Posterior 也是 Gaussian

$$\therefore p(w|X,Y) \propto p(Y|w,X)\,p(w)$$

$$\propto \prod_{i=1}^{N} N(y_i|w^T x_i, \sigma^2)\cdot N(w|m_0,S_0)$$

$$p(Y|w,X) = \prod_{i=1}^{N} \frac{1}{(2\pi)^{\frac{1}{2}}\sigma} e^{-\frac{(y_i - w^T x_i)^2}{2\sigma^2}}$$

$$= \frac{1}{(2\pi)^{\frac{N}{2}}\sigma^N} e^{-\frac{\sum_{i=1}^{N}(y_i - w^T x_i)^2}{2\sigma^2}}$$

$$= C \cdot e^{-\frac{(Y-XW)^T(Y-XW)}{2\sigma^2}}$$

$$p(W \mid X, Y) \propto e^{-\frac{(Y-XW)^T(Y-XW)}{2\sigma^2}} \, e^{-\frac{(W-m_0)^T(W-m_0)}{2S_0}}$$

$$\propto e^{\left(-\frac{1}{2\sigma^2}(Y^T - X^T W^T)(Y-XW) - \frac{1}{2} S_0^{-1}(W^T - m_0^T)(W-m_0)\right)}$$

$$= e^{\left(-\frac{1}{2\sigma^2}(Y^TY - 2Y^TXW + W^TX^TXW) - \frac{1}{2}(W^TS_0^{-1}W - 2W m_0 S_0^{-1} + m_0^T m_0 S_0^{-1})\right)}$$

$$= N(W \mid m_N, S_N)$$

二次項 : $-\frac{1}{2\sigma^2} W^T X^T X W - \frac{1}{2} W^T S_0^{-1} W$

$$= -\frac{1}{2} W^T \underbrace{(\alpha^{-2} X^T X + S_0^{-1})}_{= S_N^{-1} = A} W$$

一次項 : $-\frac{1}{2\sigma^2}(-2Y^TXW) - \frac{1}{2}(-2W^T m_0 S_0^{-1})$

$$= \sigma^{-2} Y^T X W + W m_0 S_0^{-1}$$

$$= \underbrace{(\sigma^{-2} Y^T X + m_0 S_0^{-1})}_{= m_N^T A = m_N^T S_N^{-1} = \sigma^{-2} Y^T X + m_0 S_0^{-1}} W$$

$$\Rightarrow S_N^{-1} m_N = \sigma^{-2} X^T Y + m_0 S_0^{-1}$$

$$\Rightarrow m_N = S_N (m_0 S_0^{-1} + \sigma^{-2} X^T Y)$$

根據課本的定義：$\sigma^2 = \beta^{-1}$

$X = \phi$

$Y = t$

故 $m_N = S_N(m_0 S_0^{-1} + \beta\phi^T t)$

$S_N^{-1} = S_0^{-1} + \beta\phi^T\phi$ #

$$p(x) = \mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

指數部分展開：

$$-\frac{1}{2}(x^T\Sigma^{-1} - \mu^T\Sigma^{-1})(x-\mu)$$

$$= -\frac{1}{2}(x^T\Sigma^{-1}x - 2\mu^T\Sigma^{-1}x + \Delta)$$

二次項可觀察到Σ

一次項可觀察到μ

圖 4、Bayesian Linear Regression 推導

根據以上推導出的公式，並依照課本的 3.52 設定m_0為 0，而S_0中的α則設定成1，noise precision parameter β的初值設定為$1/0.001$；透過以上敘述，將 Bayesian linear regression 寫成一個類別 BLR()，如圖 5 所示，而在測試集上計算 MSE 的結果如圖 6。

```python
class BLR:
    """ Bayesian linear regression
    Args:
        prior_mean: Mean values of the prior distribution (m_0)
        prior_cov: Covariance matrix of the prior distribution (S_0)
        noise_var: Variance of the noise distribution
    """

    def __init__(self, prior_mean: np.ndarray, prior_cov: np.ndarray, noise_var: float):
        self.prior_mean = prior_mean[:, np.newaxis]  # column vector of shape (d, 1)
        self.prior_cov = prior_cov  # matrix of shape (d, d)

        # We also know the variance of the noise
        self.noise_var = noise_var  # single float value
        self.noise_precision = 1 / noise_var

        # Before performing any inference the posterior mean and covariance equal the prior mean and variance
        self.post_mean = self.prior_mean  # corresponds to m_N in formulas
        self.post_cov = self.prior_cov  # corresponds to S_N in formulas

    def fit(self, x: np.ndarray, y: np.ndarray):
        y = y[:, np.newaxis]

        # (3.51)
        design_matrix = self.compute_design_matrix(x)

        design_matrix_dot_product = design_matrix.T @ design_matrix
        inv_prior_cov = np.linalg.inv(self.prior_cov)
        self.post_cov = np.linalg.inv(inv_prior_cov + self.noise_precision * design_matrix_dot_product)  # Eq. (3.51) S_N

        # (3.50)
        self.post_mean = self.post_cov @ (inv_prior_cov @ self.prior_mean + self.noise_precision * design_matrix.T @ y) # m_N

        return self.post_mean, self.post_cov

    def compute_design_matrix(self, features: np.ndarray) -> np.ndarray:
        n_samples = len(features)
        phi_0 = np.ones(n_samples).reshape(-1, 1)
        design_matrix = np.concatenate([phi_0, features], axis=1)
        return design_matrix

    def predict(self, x: np.ndarray):
        design_matrix = self.compute_design_matrix(x)

        pred_mean = design_matrix.dot(self.post_mean)

        return pred_mean
```

圖 5、Bayesian Linear Regression

```
(BLR)Testing set's MSE : 128.95446853877
```

圖 6、BLR 在測試集上的 MSE 結果

3. the difference between MLR and BLR

從理論上做比較，MLR 假設參數是固定的但未知的數值，所以迴歸任務是找到 Maximum Likelihood of the data 的參數值，而 BLR 假設參數本身就是隨機的，並且具有某種 prior distribution，所以當獲得新的數據時，就可以透過已知的 prior，得知 posterior；而從模型複雜度與 overfitting 的角度出發，MLR 容易導致過擬合，特別是在數據量小或者模型複雜度高的情況下，而因為 BLR 事先得知 prior，所以可以實現對模型複雜度的控制，從而對過擬合具有一定的抵抗性。表 1 將 BLR 的β設定為$1/0.001$，α則設定成1，特徵選擇'Age'、'Height'、'Weight'、'Duration'、'Heart_Rate'、'Body_Temp'，並調整訓練集的數量，其餘的皆當作測試

集，觀察訓練集數量與 MSE 之間的關係，從表 1 可以得知當特徵數量越多時，代表模型複雜度越高，故所需要的資料數量就越多，否則 MLR 就無法學習到正確的權重，而因為 BLR 事先得知 prior，故對過擬合的抑制較高。

訓練集數量	MSE	
	MLR	BLR
3	3216957722.0427	710.4327
5	5247.7216	704.5537
10	632.4809	483.4831
20	255.6936	279.3450
100	137.2844	137.1086
500	131.5810	131.4173
1000	128.8590	128.8252
5000	129.7301	129.7280
10000	131.2903	131.2886

表 1、訓練集數量與 MSE 之間的關係

從圖 4 的推導及課本的 3.53、3.54 中可以得知，在做 BLR 時需要調整α、β等參數，故設定β為$1/100$，α為20，選擇'Duration'作為特徵，使模型的不確定性增加，同時觀察訓練集的數量對 BLR 結果的影響，使用 MLR 作為對照組，並以 7:1:2 的比例來劃分訓練、驗證和測試集；在圖 7 中，當 BLR 只用 10 筆資料進行擬合時，可以觀察到擬合結果的變化幅度較大，這反映出模型的不確定性增加；而圖 8 將 BLR 的訓練資料增加至 10000 筆時，結果與使用 MLR 的結果變得幾乎一致，這表明隨著訓練數據的增加，BLR 的不確定性減少，模型的穩定性提高，並且能夠更準確地擬合數據。

然而，每次迭代都需遍歷整個訓練數據多次，如果將訓練數據載入內存，可能限制數據大小；如果不將數據載入內存，則反覆讀寫訓練數據消耗大量時間；而

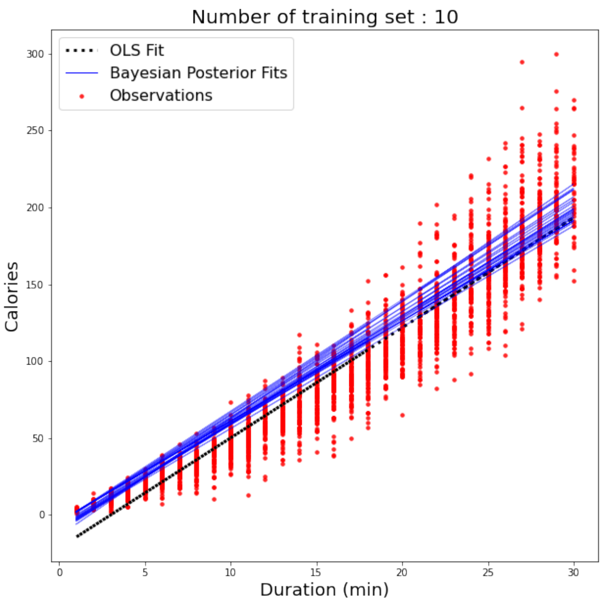

圖 7、10 筆資料

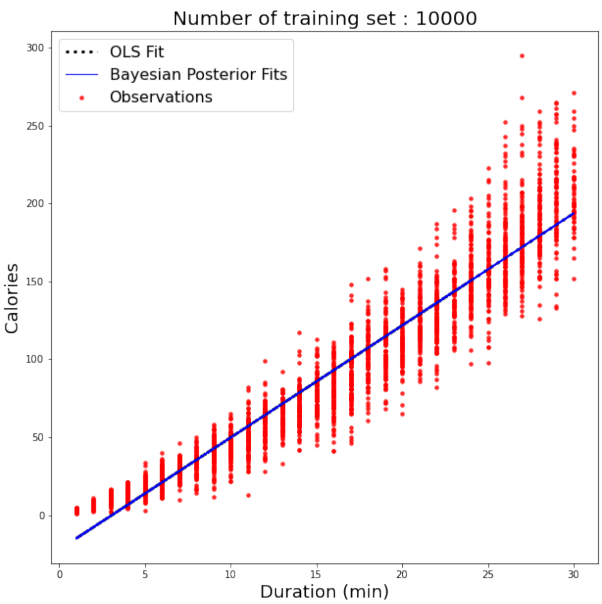

圖 8、10000 筆資料

4. best model

Gradient Boosting Decision Tree(GBDT)的主要思想是利用弱分類器(決策樹)進行迭代訓練，以得到最優模型。這種方法具有訓練效果好、不易過擬合等優點。然而，每次迭代都需遍歷整個訓練數據多次，如果將訓練數據載入內存，可能限制數據大小；如果不將數據載入內存，則反覆讀寫訓練數據消耗大量時間；而 LightGBM 是針對大數據和效率需求設計，採用兩種創新技術：Gradient-based

One-Side Sampling(GOSS)和 Exclusive Feature Bundling(EFB)，使大數據環境下的運算更高效；GOSS 是一種數據採樣策略，保留梯度大(即錯誤大)的資料，並對梯度小的資料進行隨機採樣，提升運算速度並保留大部分重要資訊。EFB 是一種特徵合併策略，能捆綁大數據中的稀疏特徵，減少空間使用，進一步提高計算速度。綜合來說，LightGBM 解決了大數據環境下的機器學習問題，它繼承了 GBDT 的優點，並在效率和大數據處理能力上實現重大優化和提升。

LightGBM 中樹的數目設定成 3000，並設定 early stopping 與 L2 正規化以防止 overfitting，而在測試集上計算 MSE 的結果如圖 9。

```
(lightgbm)Testing set's MSE : 1.7108901029512884
```

圖 9、LightGBM 在測試集上的 MSE 結果

參考文獻

[1] https://ithelp.ithome.com.tw/articles/10231807

[2] https://medium.com/@alexm5492/linear-regression-from-scratch-3-methods-fa4211f445a0

[3] https://alpopkes.com/posts/machine_learning/bayesian_linear_regression/

[4] https://www.cnblogs.com/nxf-rabbit75/p/10382368.html

[5] https://cedar.buffalo.edu/~srihari/CSE574/Chap3/3.4-BayesianRegression.pdf

[6] https://openai.com/blog/chatgpt

[7] https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html