CS 182: Introduction to Machine Learning, Fall 2022
Homework 2

(Due on Wednesday, Oct. 26 at 11:59pm (CST))

Notice:

- Please submit your assignments via Gradescope. The entry code is G2V63D.
- Please make sure you select your answer to the corresponding question when submitting your assignments.
- Each person has a total of five days to be late without penalty for all the assignments. Each late delivery less than one day will be counted as one day.

1. [20 points] [*Bayesian Decision Theory*]

   (a) Suppose that in an experiment 10,000 in total Salmon and Sea bass were observed and recorded their features, one of which is the lightness of skin. Recorders evaluated the lightness by 10-point criterion. If the lightness value is larger or equal to 5 points, it is labelled as "light"; otherwise "dark". Here gives the table recording the amount of fish in different classes and lightness.

	C_1: Salmon	C_2: Sea bass
$x =$ light	2,125	1,000
$x =$ dark	6,375	500

Table 1: The number of fish in different classes and lightness

   Specify decision rules via likelihood and posterior respectively, and give examples with $x =$ light to use the rules. [10 points]

   (b) For a K-class classification problem, the loss matrix $[\lambda_{ik}]_{i,k=1}^{K}$ records loss values for misclassification. Specifically, λ_{ik} is the loss for classifying a datapoint belonging the class C_k into C_i. And the loss incurred for selecting the reject option with threshold θ is λ. For an input $\mathbf{x}$, if $P(C_k|\mathbf{x}) \leq \theta$ (with $0 \leq \theta \leq 1$), the action of classifying $\mathbf{x}$ into C_k is rejected, even if $\mathbf{x}$ may actually belong to C_k.

   (i) Find the optimal decision rule that will give the minimum expected loss. [5 points]

   (ii) If the loss matrix is given by $[\lambda_{ik}]_{i,k=1}^{K} = \mathbf{1} - \mathbf{I}$, where $\mathbf{1}$ and $\mathbf{I}$ denote the all-one matrix and the identity respectively, and we set $0 \leq \lambda \leq 1$. State the relationship between λ and the rejection threshold θ. [5 points]

Solution:

(a) *Decision rule by likelihood*: If $P(x|C_1) > P(x|C_2)$, choose C_1; otherwise C_2.
Example: Sea bass is more likely to be light, because $P(x = \text{light}|C_1) = \frac{2125}{2125+6375} = 0.25$ which is less than $P(x = \text{light}|C_2) = \frac{1000}{1000+500} = 0.67$. So I prefer to classify a light fish as Sea bass.
Decision rule by posterior: If $P(C_1|x) > P(C_2|x)$, choose C_1; otherwise C_2.
Example: If I have already caught a light fish, I would classify it into Salmon. Because $P(C_1|x = \text{light}) = \frac{2125}{2125+1000} = 0.68$ which is larger than $P(C_1|x = \text{light}) = \frac{1000}{2125+1000} = 0.32$.
(P.S.: Computing posterior by the Bayes Formula is recommended, and you can think about the cause of different classification with the same feature $x =$ light.)

(b) (i) Suppose that an input $\mathbf{x}$ belongs to class C_k with probability $P(C_k|\mathbf{x})$. If we misclassify it to class C_i, we will incur an expected loss of $\sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x})$, whereas if we select the reject option we will incur a loss of λ. Thus, if $i^* = \arg\min_i \sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x})$, then we minimize the expected loss if we take the following action:

$$\text{choose} \begin{cases} C_{i^*}, & \text{if } \min_i \sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x}) < \lambda, \\ \text{reject}, & \text{otherwise.} \end{cases}$$

(ii) When the loss matrix is $[\lambda_{ik}]_{i,k=1}^{K} = \mathbf{1} - \mathbf{I}$, we have

$$\sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x}) = 1 - P(C_i|\mathbf{x}).$$

If $1 - P(C_i|\mathbf{x}) < \lambda$, or equivalently, $P(C_i|\mathbf{x}) < 1 - \lambda$, we reject the action of classifying $\mathbf{x}$ into class C_i. Thus, $\theta = 1 - \lambda$.

2. [20 points] [*Parameter Estimation*] Given a a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we assume that feature $\mathbf{x}_i \in \mathbb{R}^p$ and label $y_i \in \mathbb{R}$ are related via the equation

$$y_i = \mathbf{w}^T \mathbf{x}_i + e_i, \qquad i = 1, \ldots, n$$

where $\mathbf{w}$ is the parameter to be learned and e_i is an error term.

(a) Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ and $\mathbf{y} = [y_1, \ldots, y_n]^T$. Assume that $e_i \sim \mathcal{N}(0, \sigma^2)$ and $\mathbf{X}^T\mathbf{X}$ is a full-rank matrix. Derive the maximum likelihood estimate $\mathbf{w}_{ML} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. [10 points]

(b) Assume $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \nu^2\mathbf{I})$. Derive the maximum a posteriori estimate $\mathbf{w}_{MAP} = (\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\nu^2}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$. [10 points]

Solution:

(a) Since $e_i \sim \mathcal{N}(0, \sigma^2)$, we have $y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2)$. The log likelihood function is

$$L(\mathbf{w}) = \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}\right)\right)$$

Setting $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = 0$ yields $\mathbf{w}_{ML} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

(b) By the Bayes' theorem, the posteriori distribution of $\mathbf{w}$ is

$$\begin{aligned} p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}; \nu, \sigma) &= \frac{p(\mathbf{w}, \mathbf{y} \mid \mathbf{X}; \nu, \sigma)}{\sum_w p(\mathbf{w}, \mathbf{y} \mid \mathbf{X}; \nu, \sigma)} \\ &\propto p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}; \sigma) p(\mathbf{w} \mid \nu) \end{aligned}$$

The maximum a posteriori estimate

$$\begin{aligned} \mathbf{w}_{MAP} &= \arg\max p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}; \nu, \sigma) \\ &= \arg\max p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}; \sigma) p(\mathbf{w} \mid \nu) \\ &= \arg\max \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}; \sigma) + \log p(\mathbf{w} \mid \nu) \\ &= \arg\max -\frac{1}{2\sigma^2} \left\|\mathbf{y} - \mathbf{w}^T\mathbf{X}\right\|^2 - \frac{1}{2\nu^2}\mathbf{w}^T\mathbf{w} \\ &= (\mathbf{X}^T\mathbf{X} + \frac{\sigma^2}{\nu^2}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

3. [20 points] [*Multilayer Perceptrons*] Consider a multi-class classification problem with L training samples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_L, \mathbf{y}_L)$, where the input $\mathbf{x}_l \in \mathbb{R}^N$ contains N features and output $\mathbf{y}_l \in \mathbb{R}^M$ is a zero vector with one entry equals one to indicate the class of sample l, $l = 1, \ldots, L$. Suppose we use a two-layer neural network following the batch learning scheme, then the loss function can be defined as

$$\ell\left(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\right) = -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\text{softmax}\left(\mathbf{W}_2 \text{sigmoid}\left(\mathbf{W}_1 \mathbf{x}_l + \mathbf{b}_1\right) + \mathbf{b}_2\right)\right),$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are the training parameters. Derive the update rule for all parameters using gradient descent.

Solution 1:

Rewrite the loss function as

$$\ell\left(\mathbf{K}, \mathbf{W}, \mathbf{c}, \mathbf{b}\right) = -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\text{softmax}\left(\mathbf{W} \text{sigmoid}\left(\mathbf{K}\mathbf{x}_l + \mathbf{c}\right) + \mathbf{b}\right)\right),$$

the original problem is equivalent to derive update rules for $\mathbf{K}$, $\mathbf{W}$, $\mathbf{c}$, and $\mathbf{d}$.

First, we need to compute gradients over them.

A criterion to check your gradient formula is

$$\dim\left(\frac{\mathsf{dloss}}{\mathsf{dparameter}}\right) = \dim(\text{parameter}).$$

Suppose $\mathbf{K} = \begin{bmatrix} \mathbf{k}_1^T \\ \vdots \\ \mathbf{k}_D^T \end{bmatrix} = \begin{bmatrix} k_{11} & \cdots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{D1} & \cdots & k_{DN} \end{bmatrix} \in \mathbb{R}^{D \times N}$, $\mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_D \end{bmatrix} \in \mathbb{R}^D$, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_M^T \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1D} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{MD} \end{bmatrix} \in \mathbb{R}^{M \times D}$, and $\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_M \end{bmatrix} \in \mathbb{R}^M$. Denote the sigmoid function as $\sigma(\cdot)$. Then, $\mathbf{a}_l = \mathbf{K}\mathbf{x}_l + \mathbf{c} = \begin{bmatrix} \mathbf{k}_1^T \mathbf{x}_l + c_1 \\ \vdots \\ \mathbf{k}_D^T \mathbf{x}_l + c_D \end{bmatrix} = \begin{bmatrix} a_{l,1} \\ \vdots \\ a_{l,D} \end{bmatrix} \in \mathbb{R}^D$, $\mathbf{h}_l = \sigma(\mathbf{a}_l) = \begin{bmatrix} \frac{1}{1+\exp(-a_{l,1})} \\ \vdots \\ \frac{1}{1+\exp(-a_{l,D})} \end{bmatrix} = \begin{bmatrix} h_{l,1} \\ \vdots \\ h_{l,D} \end{bmatrix} \in \mathbb{R}^D$, and $\mathbf{f}_l = \mathbf{W}\mathbf{h}_l + \mathbf{b} = \begin{bmatrix} \mathbf{w}_1^T \mathbf{h}_l + b_1 \\ \vdots \\ \mathbf{w}_M^T \mathbf{h}_l + b_M \end{bmatrix} = \begin{bmatrix} f_{l,1} \\ \vdots \\ f_{l,M} \end{bmatrix} \in \mathbb{R}^M$. We can get

$$\begin{aligned}
\ell &= -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\text{softmax}\left(\mathbf{f}_l\right)\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\begin{bmatrix} \frac{\exp(f_1)}{\mathbf{1}^T \exp(\mathbf{f}_l)} \\ \vdots \\ \frac{\exp(f_M)}{\mathbf{1}^T \exp(\mathbf{f}_l)} \end{bmatrix}\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \begin{bmatrix} f_1 - \log(\mathbf{1}^T \exp(\mathbf{f}_l)) \\ \vdots \\ f_M - \log(\mathbf{1}^T \exp(\mathbf{f}_l)) \end{bmatrix} \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \left(\mathbf{f}_l - \log(\mathbf{1}^T \exp(\mathbf{f}_l))\mathbf{1}\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{f}_l + \sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{1} \log\left(\mathbf{1}^T \exp(\mathbf{f}_l)\right).
\end{aligned}$$

Denote $\text{term\_1} = -\sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{f}_l$ and $\text{term\_2} = \sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{1} \log\left(\mathbf{1}^T \exp(\mathbf{f}_l)\right)$. The differential of ℓ is

$$\mathsf{d}\ell = \mathsf{d}\text{term\_1} + \mathsf{d}\text{term\_2},$$

where

$$\begin{aligned}
\mathsf{d}\text{term_1} &= -\sum_{l=1}^{L} \mathsf{d}\left(\mathbf{y}_l^T \mathbf{f}_l\right) \\
&= -\sum_{l=1}^{L} \mathsf{d}\left(\sum_{m=1}^{M} y_{l,m} f_{l,m}\right) \\
&= -\sum_{l=1}^{L} \sum_{m=1}^{M} y_{l,m} \mathsf{d} f_{l,m} \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \mathsf{d}\mathbf{f}_l
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{d}\text{term_2} &= \sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{1} \cdot \mathsf{d}\left(\log\left(\mathbf{1}^T \exp(\mathbf{f}_l)\right)\right) \\
&= \sum_{l=1}^{L} \frac{\mathbf{y}_l^T \mathbf{1}}{\mathbf{1}^T \exp(\mathbf{f}_l)} \cdot \mathsf{d}\left(\mathbf{1}^T \exp(\mathbf{f}_l)\right) \\
&= \sum_{l=1}^{L} \frac{\mathbf{y}_l^T \mathbf{1}}{\mathbf{1}^T \exp(\mathbf{f}_l)} \cdot \left(\sum_{m=1}^{M} \mathsf{d} \exp(f_{l,m})\right) \\
&= \sum_{l=1}^{L} \frac{\mathbf{y}_l^T \mathbf{1}}{\mathbf{1}^T \exp(\mathbf{f}_l)} \cdot \left(\sum_{m=1}^{M} \exp(f_{l,m}) \cdot \mathsf{d} f_{l,m}\right) \\
&= \sum_{l=1}^{L} \frac{\mathbf{y}_l^T \mathbf{1}}{\mathbf{1}^T \exp(\mathbf{f}_l)} \cdot \begin{bmatrix} \exp(f_{l,1}) & \cdots & \exp(f_{l,M}) \end{bmatrix} \begin{bmatrix} \mathsf{d} f_{l,1} \\ \vdots \\ \mathsf{d} f_{l,M} \end{bmatrix} \\
&= \sum_{l=1}^{L} \frac{\mathbf{y}_l^T \mathbf{1}}{\mathbf{1}^T \exp(\mathbf{f}_l)} \cdot \exp(\mathbf{f}_l)^T \mathsf{d}\mathbf{f}_l \\
&= \sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{1} \mathrm{softmax}(\mathbf{f}_l)^T \mathsf{d}\mathbf{f}_l.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathsf{d}\ell &= \mathsf{d}\text{term_1} + \mathsf{d}\text{term_2} \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \mathsf{d}\mathbf{f}_l + \sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{1} \mathrm{softmax}(\mathbf{f}_l)^T \mathsf{d}\mathbf{f}_l \\
&= \sum_{l=1}^{L} \left(\mathbf{y}_l^T \mathbf{1} \mathrm{softmax}(\mathbf{f}_l) - \mathbf{y}_l\right)^T \mathsf{d}\mathbf{f}_l \\
&= \sum_{l=1}^{L} \mathbf{v}^T \mathsf{d}\left(\mathbf{W}\mathbf{h}_l + \mathbf{b}\right) \\
&= \sum_{l=1}^{L} \mathbf{v}^T \left((\mathsf{d}\mathbf{W})\mathbf{h}_l + \mathbf{W}\mathsf{d}\mathbf{h}_l + \mathsf{d}\mathbf{b}\right) \\
&= \sum_{l=1}^{L} \mathbf{v}^T \left(\begin{bmatrix} \sum_{d=1}^{D} (\mathsf{d}w_{1d}) h_{l,d} \\ \vdots \\ \sum_{d=1}^{D} (\mathsf{d}w_{Md}) h_{l,d} \end{bmatrix} + \begin{bmatrix} \sum_{d=1}^{D} w_{1d} \mathsf{d}h_{l,d} \\ \vdots \\ \sum_{d=1}^{D} w_{Md} \mathsf{d}h_{l,d} \end{bmatrix} + \begin{bmatrix} \mathsf{d}b_1 \\ \vdots \\ \mathsf{d}b_M \end{bmatrix}\right) \\
&= \sum_{l=1}^{L} \left(\sum_{m=1}^{M} \sum_{d=1}^{D} v_m h_{l,d} \mathsf{d}w_{md} + \sum_{m=1}^{M} v_m \sum_{d=1}^{D} w_{md} \mathsf{d}h_{l,d} + \sum_{m=1}^{M} v_m \mathsf{d}b_m\right),
\end{aligned}$$

where $\mathbf{v} = \mathbf{y}_l^T \mathbf{1}\text{softmax}(\mathbf{f}_l) - \mathbf{y}_l \in \mathbb{R}^M$. Then,

$$\frac{\mathsf{d}\ell}{\mathsf{d}w_{md}} = \sum_{l=1}^{L} v_m h_{l,d} \tag{1}$$

$$\frac{\mathsf{d}\ell}{\mathsf{d}b_m} = L v_m \tag{2}$$

$$\frac{\mathsf{d}\ell}{\mathsf{d}h_{l,d}} = \sum_{m=1}^{M} v_m w_{md}. \tag{3}$$

Moreover, by the chain rule,

$$\frac{\mathsf{d}\ell}{\mathsf{d}k_{dn}} = \sum_{l=1}^{L} \frac{\mathsf{d}\ell}{\mathsf{d}h_{l,d}} \frac{\mathsf{d}h_{l,d}}{\mathsf{d}a_{l,d}} \frac{\mathsf{d}a_{l,d}}{\mathsf{d}k_{dn}},$$

where

$$\frac{\mathsf{d}h_{l,d}}{\mathsf{d}a_{l,d}} = \frac{\mathsf{d}\sigma(a_{l,d})}{\mathsf{d}a_{l,d}} = \sigma(a_{l,d})(1 - \sigma(a_{l,d}))$$

and

$$\frac{\mathsf{d}a_d}{\mathsf{d}k_{dn}} = \frac{\mathsf{d}(\mathbf{k}_d^T \mathbf{x}_l + c_d)}{\mathsf{d}k_{dn}} = x_{l,n}.$$

So

$$\frac{\mathsf{d}\ell}{\mathsf{d}k_{dn}} = \sum_{l=1}^{L} \sum_{m=1}^{M} v_m w_{md} \sigma(a_{l,d})(1 - \sigma(a_{l,d})) x_{l,n}. \tag{4}$$

Similarly, by the chain rule,

$$\frac{\mathsf{d}\ell}{\mathsf{d}c_d} = \sum_{l=1}^{L} \frac{\mathsf{d}\ell}{\mathsf{d}h_{l,d}} \frac{\mathsf{d}h_{l,d}}{\mathsf{d}a_{l,d}} \frac{\mathsf{d}a_{l,d}}{\mathsf{d}c_d},$$

where

$$\frac{\mathsf{d}\ell}{\mathsf{d}c_d} = \frac{\mathsf{d}(\mathbf{k}_d^T \mathbf{x}_l + c_d)}{\mathsf{d}c_d} = 1.$$

So

$$\frac{\mathsf{d}\ell}{\mathsf{d}c_d} = \sum_{l=1}^{L} \sum_{m=1}^{M} v_m w_{md} \sigma(a_{l,d})(1 - \sigma(a_{l,d})). \tag{5}$$

----------Matrix form----------

Rewrite (1), (2), (3), and (5) in the form of matrix as

$$\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{W}} = \sum_{l=1}^{L} \begin{bmatrix} v_1 h_{l,1} & \cdots & v_1 h_{l,D} \\ \vdots & \ddots & \vdots \\ v_M h_{l,1} & \cdots & v_M h_{l,D} \end{bmatrix} = \sum_{l=1}^{L} \mathbf{v}\mathbf{h}_l^T \in \mathbb{R}^{M \times D} \tag{6}$$

$$\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{b}} = L \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix} = L\mathbf{v} \in \mathbb{R}^M \tag{7}$$

$$\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{h}_l} = \begin{bmatrix} \sum_{m=1}^{M} v_m w_{m1} \\ \vdots \\ \sum_{m=1}^{M} v_m w_{mD} \end{bmatrix} = \mathbf{W}^T \mathbf{v} \in \mathbb{R}^D$$

$$\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{c}} = \begin{bmatrix} \sum_{l=1}^{L} \sum_{m=1}^{M} v_m w_{m1} \sigma(a_{l,1})(1 - \sigma(a_{l,1})) \\ \vdots \\ \sum_{l=1}^{L} \sum_{m=1}^{M} v_m w_{mD} \sigma(a_{l,D})(1 - \sigma(a_{l,D})) \end{bmatrix} = \sum_{l=1}^{L} \left(\mathbf{W}^T \mathbf{v}\right) \odot \sigma(\mathbf{a}_l) \odot (\mathbf{1} - \sigma(\mathbf{a}_l)) \in \mathbb{R}^D. \tag{8}$$

Although we can also stack (4) to get its matrix form, here we use the chain rule to derive it. By the chain rule

$$\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{K}} = \sum_{l=1}^{L} \frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{h}_l} \frac{\mathsf{d}\mathbf{h}_l}{\mathsf{d}\mathbf{a}_l} \frac{\mathsf{d}\mathbf{a}_l}{\mathsf{d}\mathbf{K}},$$

where

$$\begin{aligned}
\frac{\mathsf{d}\mathbf{h}_l}{\mathsf{d}\mathbf{a}_l} &= \begin{bmatrix} \frac{\mathsf{d}h_{l,1}}{\mathsf{d}a_{l,1}} & \cdots & \frac{\mathsf{d}h_{l,1}}{\mathsf{d}a_{l,D}} \\ \vdots & \ddots & \vdots \\ \frac{\mathsf{d}h_{l,D}}{\mathsf{d}a_{l,1}} & \cdots & \frac{\mathsf{d}h_{l,D}}{\mathsf{d}a_{l,D}} \end{bmatrix} \\
&= \mathrm{Diag}(\begin{bmatrix} \frac{\mathsf{d}\sigma(a_{l,1})}{\mathsf{d}a_{l,1}} \\ \vdots \\ \frac{\mathsf{d}\sigma(a_{l,D})}{\mathsf{d}a_{l,D}} \end{bmatrix}) \\
&= \mathrm{Diag}(\begin{bmatrix} \sigma(a_{l,1})(1-\sigma(a_{l,1})) \\ \vdots \\ \sigma(a_{l,D})(1-\sigma(a_{l,D})) \end{bmatrix}) \\
&= \mathrm{Diag}\left(\sigma(\mathbf{a}_l) \odot (1-\sigma(\mathbf{a}_l))\right)
\end{aligned}$$

and

$$\frac{\mathsf{d}\mathbf{a}_l}{\mathsf{d}\mathbf{K}} = \frac{\mathsf{d}(\mathbf{K}\mathbf{x}_l + \mathbf{c})}{\mathsf{d}\mathbf{K}} = \frac{(\mathsf{d}\mathbf{K})\mathbf{x}_l + \mathbf{K}\mathsf{d}\mathbf{x}_l + \mathsf{d}\mathbf{c}}{\mathsf{d}\mathbf{K}} = \mathbf{x}_l.$$

So

$$\begin{aligned}
\frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{K}} &= \sum_{l=1}^{L} \frac{\mathsf{d}\ell}{\mathsf{d}\mathbf{h}_l} \mathrm{Diag}\left(\sigma(\mathbf{a}_l) \odot (1-\sigma(\mathbf{a}_l))\right) \mathbf{x}_l \\
&= \sum_{l=1}^{L} \mathrm{Diag}\left(\mathbf{W}^T\mathbf{v} \odot \sigma(\mathbf{a}_l) \odot (1-\sigma(\mathbf{a}_l)) \odot \mathbf{x}_l\right) \in \mathbb{R}^{D\times N}. \qquad (9)
\end{aligned}$$

So update rules are

$$\begin{aligned}
\mathbf{W}^{update} &= \mathbf{W} - \alpha\frac{d\ell}{d\mathbf{W}} \\
\mathbf{b}^{update} &= \mathbf{b} - \alpha\frac{d\ell}{d\mathbf{b}} \\
\mathbf{K}^{update} &= \mathbf{K} - \alpha\frac{d\ell}{d\mathbf{K}} \\
\mathbf{c}^{update} &= \mathbf{c} - \alpha\frac{d\ell}{d\mathbf{c}},
\end{aligned}$$

where α is the learning rate.

4. [20 points] [*Bayesian Decision Theory, Linear Discrimination*]

(a) Consider a binary classification problem, with the class conditional density

$$p(\mathbf{x} \mid C_i) = \frac{1}{(2\pi)^{\frac{d}{2}} \det(\mathbf{\Sigma}_i)^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \mathbf{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right], \ i = 1, 2.$$

Assume that $\mathbf{\Sigma}_i = \sigma^2 \mathbf{I}$ for all i, and that the priors for the two classes are not equal, i.e., $P(C_1) \neq P(C_2)$. If our target is to minimize the expected loss, a.k.a. conditional risk, (as in the lecture notes, we denote by λ_{ij} the loss incurred for assigning an input $\mathbf{x}$ to class C_i when the actual state is C_j), derive the decision boundary, and explain how geometrically it differs from that when one minimizes the misclassification error, a.k.a. probability of error. [10 points]

(b) Assume that $\mathbf{x}$ is 2-dimensional, and that the two dimensions are independent following the Laplace distribution:

$$p(x_j \mid C_i) = \frac{1}{2\sigma} \exp(-\frac{|x_j - \mu_{ij}|}{\sigma}), \ j = 1, 2.$$

By minimizing the misclassification error, obtain and draw the decision boundary when $\mu_{11} = 1$, $\mu_{12} = 1$, $\mu_{21} = 3$, $\mu_{22} = 5$, $\sigma = 1$, and $P(C_1) = P(C_2)$. [10 points]

Solution:

(a) When using the loss functions in the binary classification case, we will decide ω_1 if $(\lambda_{21}-\lambda_{11})p(\mathbf{x}|C_1)P(C_1) > (\lambda_{22}-\lambda_{12})p(\mathbf{x}|C_2)P(C_2)$.Taking the logarithm and the discriminant function can be obtained by

$$\begin{aligned} g_i(\mathbf{x}) &= \ln c_i + \ln p\left(\mathbf{x} \mid C_i\right) + \ln P\left(C_i\right) \\ &= -\frac{\left\|\mathbf{x}-\boldsymbol{\mu}_i\right\|^2}{2\sigma^2} + \left[\ln c_i + \ln P\left(C_i\right) + const.\right], \end{aligned}$$

where $c_1 = (\lambda_{21} - \lambda_{11})$, $c_2 = (\lambda_{22} - \lambda_{12})$, *const.* is a constant.
By applying the result in slides, the decision boundary is a line: $\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0) = 0$, where

$$\begin{aligned} \mathbf{w} &= \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2, \\ \mathbf{x}_0 &= \frac{1}{2}\left(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2\right) - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{c_1 P\left(C_1\right)}{c_2 P\left(C_2\right)} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right). \end{aligned}$$

We know the decision boundary obtained by minimizing the error is the line $\mathbf{w}^T(\mathbf{x} - \mathbf{x}_{e0}) = 0$, where

$$\mathbf{x}_{e0} = \frac{1}{2}\left(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2\right) - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{P\left(C_1\right)}{P\left(C_2\right)} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right).$$

Since

$$\begin{aligned} \mathbf{x}_0 &= \frac{1}{2}\left(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2\right) - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{c_1 P\left(C_1\right)}{c_2 P\left(C_2\right)} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right) \\ &= \frac{1}{2}\left(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2\right) - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{P\left(C_1\right)}{P\left(C_2\right)} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right) - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{c_1}{c_2} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right) \\ &= \mathbf{x}_{e0} - \frac{\sigma^2}{\left\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right\|^2} \ln \frac{c_1}{c_2} \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right), \end{aligned}$$

The decision boundary obtained by minimizing the loss is a translation of its error minimization counterpart.

(b) When minimize error, the discriminant functions are

$$g_i(\mathbf{x}) = -\ln(2\sigma) - \frac{\|\mathbf{x} - \boldsymbol{\mu}_i\|_1}{\sigma} + \ln P(C_i), \quad i = 1, 2.$$

The decision boundary satisfies the equation

$$g_1(\mathbf{x}) - g_2(\mathbf{x}) = 0.$$

That is,

$$\|\mathbf{x} - \boldsymbol{\mu}_1\|_1 - \|\mathbf{x} - \boldsymbol{\mu}_2\|_1 - \sigma \ln \frac{P\left(C_1\right)}{P\left(C_2\right)} = 0$$

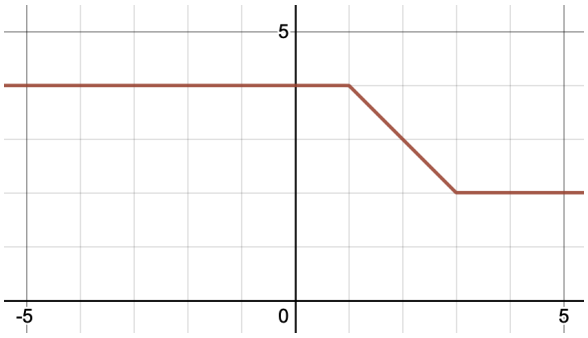
Using those values, the boundary satisfies

$$(|x_1 - 1| - |x_1 - 3|) + (|x_2 - 1| - |x_2 - 5|) = 0.$$

The decision boundary is

$$\begin{cases} x_1 + x_2 = 5, & 1 \leq x_1 \leq 3, \\ x_2 = 2, & x_1 > 3, \\ x_2 = 4, & x_1 < 1. \end{cases}$$

The following figure shows the decision boundary



5. [20 points] [*Coding: Logistic Regression*] Complete "HW2-Coding.ipynb". After completion, you should convert your notebook to PDF, and concatenate the writing part and the coding part into one PDF which is the file to submit.

Homework 2 Coding: Logistic Regression.

Please export this jupyter notebook as PDF, and hand in .pdf (with writing part) file.

In this part of homework, you need to implement Logistic Regression using Python in this jupyter notebook.

Part 0: Preparation before training.

This part loads the necessary libraries and dataset. You are only required to do the normalization by yourself.

In [26]:

```
#import all the required libraries. You need to implement them in first.
import pandas as pd
from sklearn.datasets import load_breast_cancer
import numpy as np
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [27]:

```
#Loading the dataset including features and binary labels
data = load_breast_cancer().data
target = load_breast_cancer().target
```

In [28]:

```
# Size of features and labels
data.shape, target.shape
```

Out[28]:

```
((569, 30), (569,))
```

In [29]:

```
#Splitting the data into train and test sets 2:1 with certain random seed.
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.33, random_state = 42)
```

Normalizing data (by yourself)

In [30]:

```
# A useful trick before training is to normalize all features to have mean 0 and unit variance first
# Please implement this by yourself rather than use sklearn.preprocessing.StandardScaler as the comm

mean, var = np.mean(X_train,axis=0), np.var(X_train,axis=0)
m1, n1 = X_train.shape
m2, n2 =X_test.shape
for i in range(n1):
    var[i] = pow(var[i], 0.5)

for i in range(m1):
    for j in range(n1):
        X_train[i][j] = (X_train[i][j] - mean[j]) / var[j]

for i in range(m2):
    for j in range(n2):
        X_test[i][j] = (X_test[i][j] - mean[j]) / var[j]
```

Some helper functions are given below, you are free to use them or not in parts below.

In [31]:

```
# Function to predict y of x with current weights
def predict(x,w):
    y_pred=[]
    for i in range(len(x)):
        y = (np.asscalar(1/(1+np.exp(-(np.dot(w,x[i]))))))
        if y<0.5:
            y_pred.append(0)
        else:
            y_pred.append(1)
    return np.array(y_pred)
```

In [32]:

```
#Function to calulate TPR,FPR,TNR and FNR to be included in confusion matrix
def find_rates(mat):
    mat2=[]
    mat2.append((mat[0,0]))
    mat2.append((mat[1,0]))
    mat2.append((mat[0,1]))
    mat2.append((mat[1,1]))
    mat2=np.reshape(mat2, (2,2))
    mat2 = pd.DataFrame(mat2,columns=[0,1],index=[0,1])
    mat2.index.name = 'Predicted'
    mat2.columns.name = 'Actual'
    return mat2
```

Part 1: Implement Logistic Regression using sklearn.

In this part, you are firstly given an example Sklearn implementation of logistic regression. Play with them and then you should:

1. Explain the parameters and their effects in LogisticRegression().

2. Try different settings of parameters and show its performance as the example.

You can read official document from [https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

In [33]:

```
#Logistic regression using sklearn
LRexample = LogisticRegression(penalty = 'l2', C=0.1, solver = 'liblinear')
LRexample.fit(X_train, y_train)
```

Out[33]:

```
LogisticRegression(C=0.1, solver='liblinear')
```

In [34]:

```
# Predict on the test set
y_pred_sklearn = LRexample.predict(X_test)
```

In [35]:

```
# The labels of ground-truth on test set.
np.unique(y_test,return_counts=True)
```

Out[35]:

```
(array([0, 1]), array([ 67, 121], dtype=int64))
```

In [36]:

```
# The labels produced by LR model on test set.
np.unique(y_pred_sklearn,return_counts=True)
```

Out[36]:

```
(array([0, 1]), array([ 67, 121], dtype=int64))
```

In [37]:

```
# true-negative, false-negative, false-negative, true-positive
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
(tn, fp, fn, tp)
```

Out[37]:

```
(66, 1, 1, 120)
```

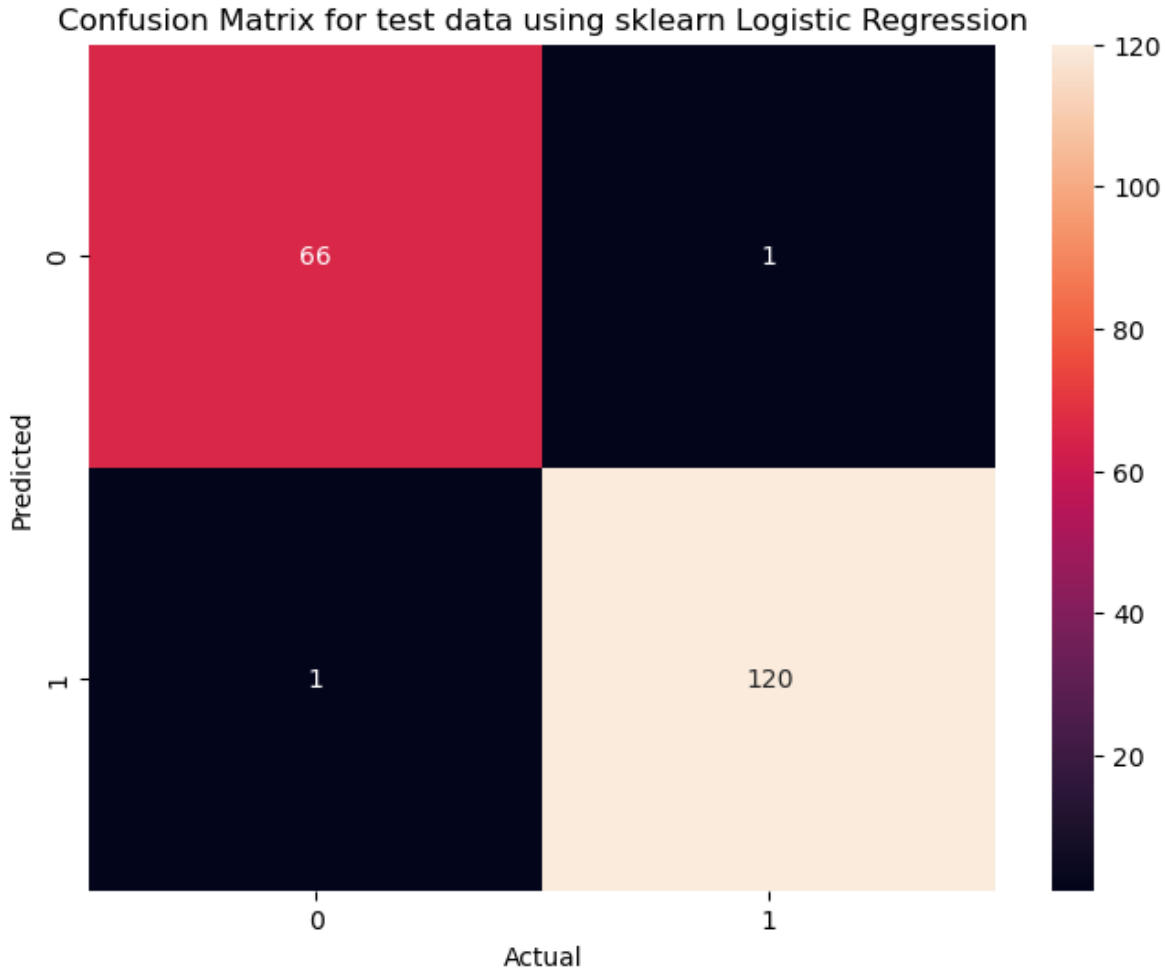
In [38]:

```
mat_test = find_rates(confusion_matrix(y_test, y_pred_sklearn))

fig=plt.figure(figsize=(8,6))
plt.title('Confusion Matrix for test data using sklearn Logistic Regression')
sns.heatmap(mat_test,annot=True,fmt='g')
```

Out[38]:

<AxesSubplot:title={'center':'Confusion Matrix for test data using sklearn Logistic Regression'}, xlabel='Actual', ylabel='Predicted'>



In [39]:

```
LRexample.score(X_test, y_test)
```

Out[39]:

0.9893617021276596

Explain the parameters and their effects in LogisticRegression( ) in the Markdown cell below.

Type *Markdown* and LaTeX: α^2

Try different settings of Sklearn implementation of logistic regression and show the performance as the example above. Write your codes below.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Part 2: Implement Logistic Regression without using its library.

In this part, you need to implement Logistic regression model using Batch Gradient Descent and Stochastic Gradient Descent by yourself. The hyperparameters of the two algorithms are given and recommended. Notice that with given hyperparameters and random seeds, the weights obtained by BGD and SGD with momentum should be unique.

Part 2.1: Implement logistic regression using Batch-GD

Decribe the Batch-GD alogrithm in the Markdown cell below. You are free to use mathematical derivation or not.

Type *Markdown* and LaTeX: α^2

In [40]:

```
"""
At each iteration, train all the samples and update weights. The initilization point should be set t
"""
n_iter=50  # number of iterations
reg=0.01   # regularization parameter lambda
r=0.1      # learning rate
sample_size=X_train.shape[0]    # batch size for BGD
N=X_train.shape[0]

w_BGD = np.zeros(np.shape(X_train)[1])
def theta(s):
    return 1 / (1 + np.exp(-s))
def gradient(X, y,_w,reg):
    gradient = np.zeros((X.shape[1]))
    for xi, yi in zip(X, y):
        gradient += np.reshape((theta(np.dot(np.transpose(_w), xi))-yi)*xi,(X.shape[1]))
    return (gradient + reg * _w) * (1 / X.shape[0])
for j in range(n_iter):
    grad = gradient( X_train, y_train, w_BGD,reg)
    w_BGD -= r * grad
```

In [41]:

```
#Getting predictions for test datapoints
y_pred_BGD = predict(X_test,w_BGD)
```

```
C:\Users\shanghaitech\AppData\Local\Temp\ipykernel_17668\330463052.py:5: Deprecation
Warning: np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead
  y = (np.asscalar(1/(1+np.exp(-(np.dot(w,x[i]))))))
```

In [42]:

```
np.unique(y_test,return_counts=True)
```

Out[42]:

```
(array([0, 1]), array([ 67, 121], dtype=int64))
```

In [43]:

```
np.unique(y_pred_BGD,return_counts=True)
```

Out[43]:

```
(array([0, 1]), array([ 71, 117], dtype=int64))
```

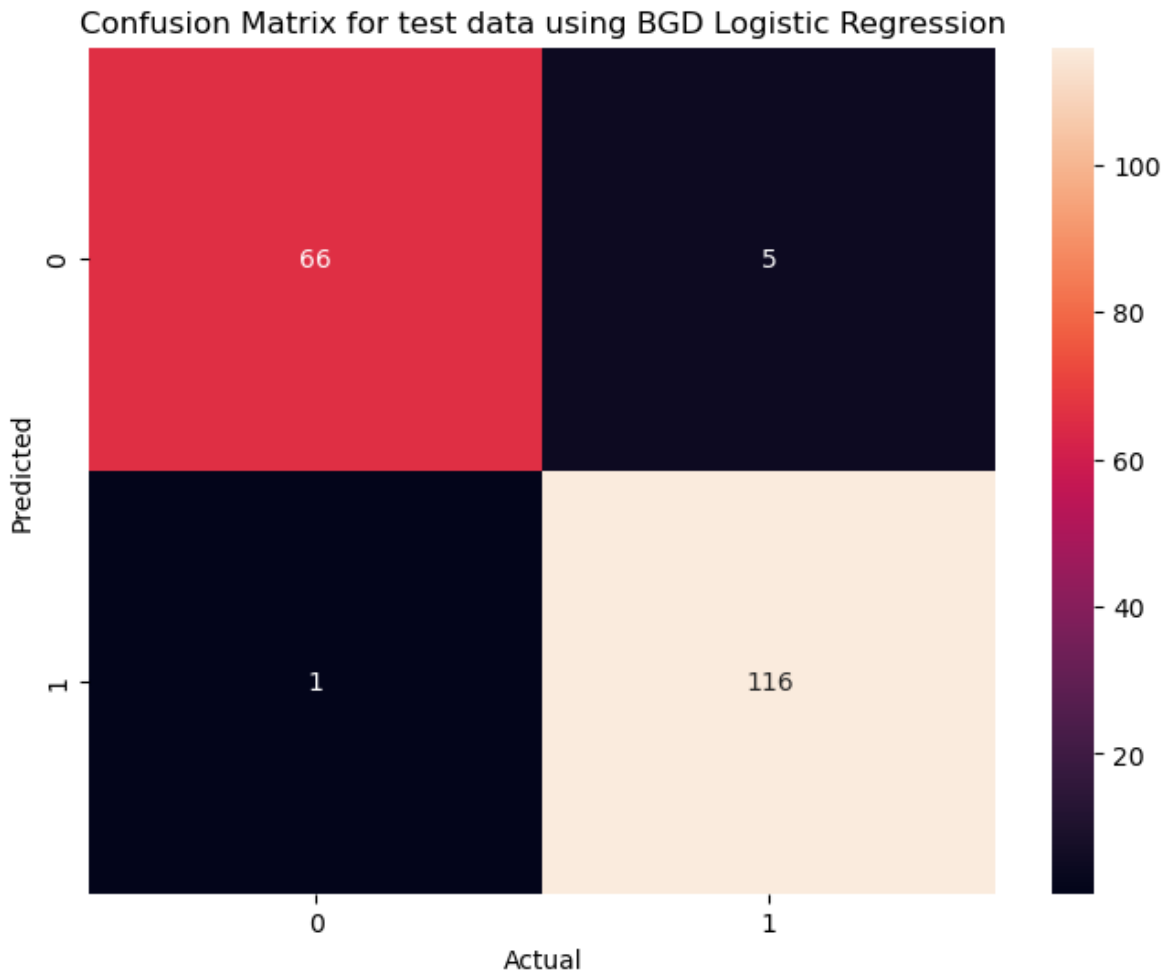
In [44]:

```
# Draw confusion matrix
mat_test = find_rates(confusion_matrix(y_test, y_pred_BGD))

fig=plt.figure(figsize=(8,6))
plt.title('Confusion Matrix for test data using BGD Logistic Regression')
sns.heatmap(mat_test,annot=True,fmt='g')
```

Out[44]:

```
<AxesSubplot:title={'center':'Confusion Matrix for test data using BGD Logistic Regr
ession'}, xlabel='Actual', ylabel='Predicted'>
```



Part 2.2: Implement logistic regression using SGD with momentum

In this part, you need to implement logistic regression using SGD method with momentum for accelerating training. Intuitively, the method tries to accelerate with keeping the 'momentum' by moving along a previous

direction. You may find Chapter 8.3 helpful https://www.deeplearningbook.org/contents/optimization.html (https://www.deeplearningbook.org/contents/optimization.html) for more details with respect to SGD, momentum and more acceleration tricks.

Decribe the SGD with momentum alogrithm in the Markdown cell below. You are free to use mathematical derivation or not.

Type *Markdown* and LaTeX: α^2

In [45]:

```
"""
At each iteration, choose 20 samples randomly and compute dJ(theta)/d(theta) among
those 20 samples then update the vector of weights with momentum. The initilization point should be

Note that the random seed at each iteration is given, do not modify it.
"""
n_iter=50  # number of iterations
reg=0.01   # regularization parameter lambda
r=0.1      # learning rate
momen = 0.5 # momentum rate
sample_size=20    # sample size for SGD
N=X_train.shape[0]

w_SGD = np.zeros(np.shape(X_train)[1])
momem = np.zeros(np.shape(X_train)[1])
for j in range(n_iter):
    np.random.seed(j)
    idx=np.random.randint(X_train.shape[0],size=sample_size)
    X_train_batch, y_train_batch = X_train[idx,:],y_train[idx]
    grad = gradient(X_train_batch, y_train_batch, w_SGD, reg)
    momem = grad + 0.5 * momem
    w_SGD -= r * momem
```

In [46]:

```
#Getting predictions for test datapoints
y_pred_SGD = predict(X_test,w_SGD)
```

```
C:\Users\shanghaitech\AppData\Local\Temp\ipykernel_17668\330463052.py:5: Deprecation
Warning: np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead
  y = (np.asscalar(1/(1+np.exp(-(np.dot(w,x[i]))))))
```

In [47]:

```
np.unique(y_test,return_counts=True)
```

Out[47]:

```
(array([0, 1]), array([ 67, 121], dtype=int64))
```

In [48]:

```
np.unique(y_pred_SGD,return_counts=True)
```

Out[48]:

```
(array([0, 1]), array([ 71, 117], dtype=int64))
```

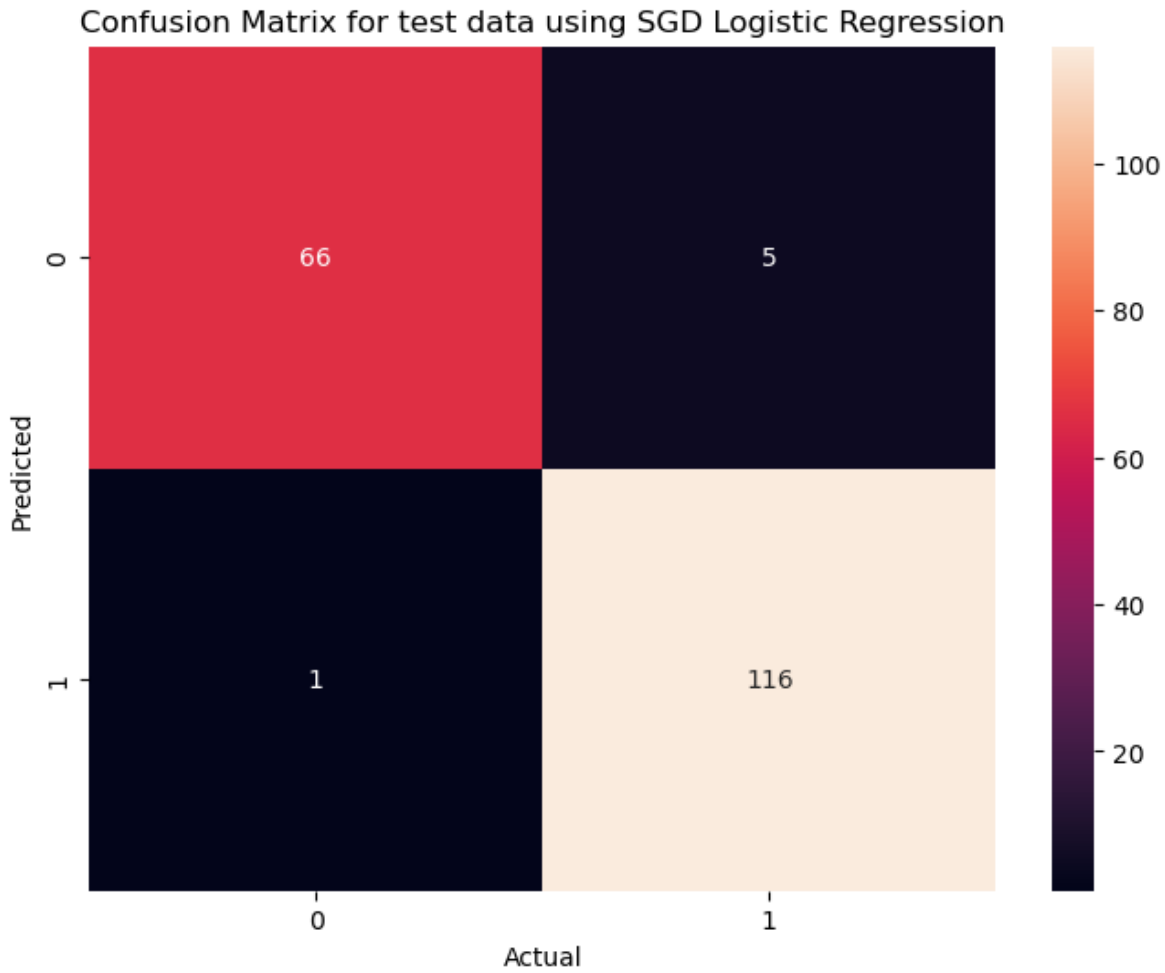
In [49]:

```
# Draw confusion matrix
mat_test = find_rates(confusion_matrix(y_test, y_pred_SGD))

fig=plt.figure(figsize=(8,6))
plt.title('Confusion Matrix for test data using SGD Logistic Regression')
sns.heatmap(mat_test,annot=True,fmt='g')
```

Out[49]:

```
<AxesSubplot:title={'center':'Confusion Matrix for test data using SGD Logistic Regr
ession'}, xlabel='Actual', ylabel='Predicted'>
```



In [50]:

```
# Print a table to show every coefficients in vector w, and compute the absolute difference between

from prettytable import PrettyTable
p = PrettyTable()
p.title='Weights from both models'
p.field_names=['SKlearn','BGD', 'SGD', 'Difference']

# You can directly run the code below to output the table or rewrite it.
# Please remain five decimal places
for i in range(30):
    p.add_row(['{:.5f}'.format(LRexample.coef_[0,i]),'{:.5f}'.format(w_BGD[i]),
               '{:.5f}'.format(w_SGD[i]), '{:.5f}'.format(abs(w_BGD[i]-w_SGD[i]))])
print(p)
```

```
+-----------------------------------------------+
|           Weights from both models            |
+----------+----------+----------+------------+
| SKlearn  |   BGD    |   SGD    | Difference |
+----------+----------+----------+------------+
| -0.33269 | -0.31320 | -0.38873 |  0.07553   |
| -0.35660 | -0.23831 | -0.35342 |  0.11511   |
| -0.32618 | -0.31280 | -0.38676 |  0.07396   |
| -0.33995 | -0.30531 | -0.38358 |  0.07827   |
| -0.12556 | -0.12162 | -0.17380 |  0.05218   |
| 0.03085  | -0.13810 | -0.12038 |  0.01772   |
| -0.37123 | -0.24994 | -0.30637 |  0.05643   |
| -0.47501 | -0.33393 | -0.43095 |  0.09701   |
| -0.04295 | -0.08672 | -0.09217 |  0.00545   |
| 0.18929  | 0.12348  | 0.23144  |  0.10796   |
| -0.45881 | -0.25481 | -0.35486 |  0.10004   |
| 0.02783  | 0.00374  | 0.00912  |  0.00538   |
| -0.35493 | -0.22650 | -0.31208 |  0.08558   |
| -0.35434 | -0.23163 | -0.32693 |  0.09530   |
| -0.07287 | -0.01296 | -0.04840 |  0.06145   |
```