Introduction to Machine Learing: Homework II

Due on Oct 26th, 2022 at 11:59pm

Professor Ziping Zhao

Bingnan Li
2020533092

1. [Bayesian Decision Theory]

(a) Specify decision rule via likelihood and posterior respectively, and give example with $x = light$ to use the rules.

Solution:

Likelihood decision rule:

$$choose \begin{cases} C = C_1,\ if\ P(x|C_1) > P(x|C_2) \\ C = C_2,\ elsewhere \end{cases}$$

Posterior decision rule:

$$choose \begin{cases} C = C_1,\ if\ P(C_1|x) > P(C_2|x) \\ C = C_2,\ elsewhere \end{cases}$$

If $x = light$, then likelihood is

$$P(x = light|C_1) = \frac{2125}{2125 + 6375} = 0.25$$

$$P(x = light|C_2) = \frac{1000}{1000 + 500} = 0.667$$

Thus, by likelihood decision rule, we choose C_2. However, the posterior probability is

$$P(C_1|x = light) = \frac{2125}{2125 + 1000} = 0.68$$

$$P(C_2|x = light) = \frac{1000}{2125 + 1000} = 0.32$$

Thus, by posterior decision rule, we choose C_1.

(b) (i) Find the optimal decision rule that will give the minimum expected loss.

Solution:

Let action α_i be classifying input $\boldsymbol{x}$ into class i and $R(\alpha_i|\boldsymbol{x})$ be the expected loss of taking action α_i, then we have

$$R(\alpha_i|\boldsymbol{x}) = \sum_{j=1}^{K} \lambda_{i,j} P(C_k|\boldsymbol{x})$$

Then the optimal decision rule is

$$\begin{cases} choose\ class\ i, & if\ R(\alpha_i|\boldsymbol{x}) = \min\{R(\alpha_j|\boldsymbol{x})\}_{j=1}^{K} < \lambda \\ reject\ \boldsymbol{x}, & if\ R(\alpha_i|\boldsymbol{x}) = \min\{R(\alpha_j|\boldsymbol{x})\}_{j=1}^{K} \geq \lambda \end{cases}$$

(ii) State the relationship between λ and the rejection threshold θ.

Solution:

Given that $\lambda_{i,k} = \begin{cases} 1, & if\ i \neq k \\ 0, & if\ i = k \end{cases}$ Then the corresponding expected loss is as following

$$R(\alpha_i|\boldsymbol{x}) = \sum_{j=1}^{K} \lambda_{i,j} P(C_k|\boldsymbol{x}) = 1 - P(C_i|\boldsymbol{x})$$

In order to get minimum expected loss, we hope:

- When $P(C_i|\boldsymbol{x}) > \theta$, we hope that $R(\alpha_i|\boldsymbol{x}) = \min\{R(\alpha_j|\boldsymbol{x})\}_{j=1}^K = 1 - P(C_i|\boldsymbol{x}) \leq \lambda$ so that we won't get expected loss any lower by manually choosing to reject $\boldsymbol{x}$.

  In order to achieve that, we need:

  $$\lambda \geq max(1 - P(C_i|\boldsymbol{x})) = 1 - \theta$$

- When $P(C_i|\boldsymbol{x}) \leq \theta$, we hope that $\lambda \leq R(\alpha_i|\boldsymbol{x}) = \min\{R(\alpha_j|\boldsymbol{x})\}_{j=1}^K = 1 - P(C_i|\boldsymbol{x})$ so that we won't get expected loss any lower by selecting to classifying $\boldsymbol{x}$ into class i.

  In order to achieve that, we need

  $$\lambda \leq min(1 - P(C_i|\boldsymbol{x})) = 1 - \theta$$

Based on that, we have

$$\lambda \geq 1 - \theta$$
$$\lambda \leq 1 - \theta$$

Therefore, we have

$$\lambda = 1 - \theta \Rightarrow \lambda + \theta = 1$$

2. [Parameter Estimation]

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

Proof:

Given that $e_i \sim \mathcal{N}(0, \sigma^2)$ and $y_i = \boldsymbol{w^Tx_i} + e_i$, we have that

$$y_i \sim \mathcal{N}(\boldsymbol{w^Tx_i}, \sigma^2)$$

Thus, the likelihood function is

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w}) = \log(P(\boldsymbol{y|X,w})) &= -\sum_{i=1}^{n}\left(\log\sqrt{2\pi}\sigma + \frac{\left(y_i - \boldsymbol{w^Tx_i}\right)^2}{2\sigma^2}\right) \\
&= -n\log\sqrt{2\pi}\sigma - \frac{\sum_{i=1}^{n}\left(y_i - \boldsymbol{w^Tx_i}\right)^2}{2\sigma^2} \\
&= -n\log\sqrt{2\pi}\sigma - \frac{||\boldsymbol{y - Xw}||_2^2}{2\sigma^2} \\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} &= -\frac{1}{2\sigma^2}\frac{\partial\boldsymbol{w^TX^TXw - 2w^TX^Ty + y^Ty}}{\partial\boldsymbol{w}} \\
&= -\frac{1}{2\sigma^2}\left(2\boldsymbol{X^TXw} - 2\boldsymbol{X^Ty}\right) = 0 \\
&\Rightarrow \boldsymbol{X^TXw = X^Ty} \\
&\Rightarrow \boldsymbol{w}_{ML} = \left(\boldsymbol{X^TX}\right)^{-1}\boldsymbol{X^Ty}
\end{aligned}$$

(b) Assume $\boldsymbol{w} \sim \mathcal{N}(0, \nu^2\boldsymbol{I})$. Derive the maximum a posteriori estimate

$$\boldsymbol{w}_{MAP} = \left(\boldsymbol{X^TX} + \frac{\sigma^2}{\nu^2}\boldsymbol{I}\right)^{-1}\boldsymbol{X^Ty}$$

Proof:

Given that $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \nu^2\boldsymbol{I})$, we know that

$$P(\boldsymbol{w}) = \frac{1}{(2\pi)^{n/2}\nu}\exp[-\frac{1}{2}(\boldsymbol{w^T}\frac{1}{\nu^2}\boldsymbol{Iw})] = \frac{1}{(2\pi)^{n/2}\nu}\exp\left(-\frac{\boldsymbol{w^Tw}}{2\nu^2}\right)$$

Then by definition, we know that

$$\begin{aligned}\boldsymbol{w}_{MAP} &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{w}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{w}} \frac{P(\mathcal{D}|\boldsymbol{w})P(\boldsymbol{w})}{P(\mathcal{D})} \\ &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}, \boldsymbol{X}|\boldsymbol{w})P(\boldsymbol{w}) \\ &= \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})P(\boldsymbol{X}|\boldsymbol{w})P(\boldsymbol{w})\end{aligned}$$

Since the value of $\boldsymbol{X}$ does not rely on $\boldsymbol{w}$, we have $P(\boldsymbol{X}|\boldsymbol{w}) = P(\boldsymbol{X})$. Sequentially, we have

$$\boldsymbol{w}_{MAP} = \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})P(\boldsymbol{X})P(\boldsymbol{w}) = \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})P(\boldsymbol{w})$$

Next, let $\mathcal{L}(\boldsymbol{w}) = \log P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})P(\boldsymbol{w}) = \log P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}) + \log P(\boldsymbol{w})$, we have

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) &= -n \log \sqrt{2\pi}\sigma - \frac{||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||_2^2}{2\sigma^2} + \log{(2\pi)^{n/2}\nu} - \frac{\boldsymbol{w^T w}}{2\nu^2} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} &= -\frac{\boldsymbol{X^T X w} - \boldsymbol{X^T y}}{\sigma^2} - \frac{\boldsymbol{w}}{\nu^2} = 0 \\ &\Rightarrow \left(\boldsymbol{X^T X} + \frac{\sigma^2}{\nu^2}\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{X^T y}\end{aligned}$$

Given that $\boldsymbol{X^T X}$ is full-rank and $\frac{\sigma^2}{\nu^2}\boldsymbol{I}$ is positive-definite, we know that $\boldsymbol{X^T X} + \frac{\sigma^2}{\nu^2}\boldsymbol{I}$ is invertible.

Therefore, we know that

$$\boldsymbol{w}_{MAP} = \left(\boldsymbol{X^T X} + \frac{\sigma^2}{\nu^2}\boldsymbol{I}\right)^{-1}\boldsymbol{X^T y}$$

3. [Parameter Estimation]

Solution:

Let $\boldsymbol{z}_l = sigmoid(\boldsymbol{W}_1\boldsymbol{x}_l + \boldsymbol{b}_1)$ and $\boldsymbol{f}_l = softmax(\boldsymbol{W_2 z}_l + \boldsymbol{b}_2)$, then

$$\begin{aligned}\ell(\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{b_1}, \boldsymbol{b_2}) &= -\sum_{l=1}^{L} \boldsymbol{y}_l^T \log \boldsymbol{f}_l \\ &= -\sum_{l=1}^{L}\sum_{k=1}^{M} y_{l,k} \log f_{l,k}\end{aligned}$$

where $f_{l,k} = softmax(\boldsymbol{w}_{2,k}^T \boldsymbol{z}_l + b_{2,k})$. Then, we have

$$\begin{aligned}\frac{\partial \ell}{\partial \boldsymbol{w_{2,i}^T z_l} + b_{2,i}} &= \frac{\partial \ell}{\partial f_{l,k}} \frac{\partial f_{l,k}}{\boldsymbol{w_{2,i}^T z_l} + b_{2,i}} \\ &= -\sum_{l=1}^{L}\sum_{k=1}^{M} \frac{y_{l,k}}{f_{l,k}} f_{l,k}(\delta_{ki} - f_{l,i}) \\ &= -\sum_{l=1}^{L}\sum_{k=1}^{M} y_{l,k}(\delta_{ki} - f_{l,i}) \\ &= -\sum_{l=1}^{L}(y_{l,i} - f_{l,i}) \\ where\ \delta_{ki} &= \begin{cases} 1, & k = i \\ 0, & k \neq i \end{cases}\end{aligned}$$

Since

$$\frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial w_{2,i,j}} = z_{l,j}$$

$$\frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial b_{2,i}} = 1$$

$$\frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial w_{1,p,q}} = \sum_{j=1}^{H} w_{2,i,j} \frac{\partial z_{l,j}}{\partial w_{1,p,q}}$$

$$\frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial b_{1,p}} = \sum_{j=1}^{H} w_{2,i,j} \frac{\partial z_{l,j}}{\partial b_{1,p}}$$

$$\frac{\partial z_{l,j}}{\partial w_{1,p,q}} = \begin{cases} z_{l,q}(1 - z_{l,q})x_{l,q},\ j = p \\ 0,\ j \neq q \end{cases}$$

$$\frac{\partial z_{l,j}}{\partial b_{1,p}} = \begin{cases} z_{l,q}(1 - z_{l,q}),\ j = p \\ 0,\ j \neq q \end{cases}$$

Where $z_{l,q} = sigmoid(\boldsymbol{w^T_{1,q} x_l} + b_{1,q})$.

Then, we have:

$$\begin{aligned} \Delta w_{2,i,j} &= -\eta \frac{\partial \ell}{\partial w_{2,i,j}} = -\eta \frac{\partial \ell}{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}} \frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial w_{2,i,j}} \\ &= \eta \sum_{l=1}^{L} (y_{l,i} - f_{l,i}) z_{l,j} \end{aligned}$$

$$\begin{aligned} \Delta b_{2,i} &= -\eta \frac{\partial \ell}{\partial b_{2,i}} = -\eta \frac{\partial \ell}{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}} \frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial b_{2,i}} \\ &= \eta \sum_{l=1}^{L} (y_{l,i} - f_{l,i}) \end{aligned}$$

$$\begin{aligned} \Delta w_{1,p,q} &= -\eta \frac{\partial \ell}{\partial w_{1,p,q}} = -\eta \sum_{i=1}^{M} \frac{\partial \ell}{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}} \frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial w_{1,p,q}} \\ &= \eta \sum_{l=1}^{L} \left[\sum_{i=1}^{M} (y_{l,i} - f_{l,i}) w_{2,i,p} \right] z_{l,p}(1 - z_{l,p}) x_{l,q} \end{aligned}$$

$$\begin{aligned} \Delta b_{1,p} &= -\eta \frac{\partial \ell}{\partial b_{1,p}} = -\eta \sum_{i=1}^{M} \frac{\partial \ell}{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}} \frac{\partial \boldsymbol{w^T_{2,i} z_l} + b_{2,i}}{\partial b_{1,p}} \\ &= \eta \sum_{l=1}^{L} \left[\sum_{i=1}^{M} (y_{l,i} - f_{l,i}) w_{2,i,p} \right] z_{l,p}(1 - z_{l,p}) \end{aligned}$$

Where

$$f_{l,i} = softmax(\boldsymbol{w}_{2,i}^T \boldsymbol{z}_l + b_{2,i})$$
$$z_{l,p} = sigmoid(\boldsymbol{w}_{1,q}^T \boldsymbol{x}_l + b_{1,q})$$

4. [Bayesian Decision Theory, Linear Discrimination]

(a) minimize the expected loss v.s. minimize the misclassification error.

Solution:

Let λ_{ij} be the loss for misclassifying $\boldsymbol{x}$ into class i while it actually belongs to class j, α_i be the action of classifying $\boldsymbol{x}$ into class i and $R(\alpha_i|\boldsymbol{x})$ be the expected loss of classifying $\boldsymbol{x}$ into class i.

Therefore,

$$R(\alpha_i|\boldsymbol{x}) = \sum_{j=1}^{2} \lambda_{ij} P(C_j|\boldsymbol{x})$$

Let $g(\boldsymbol{x}) = R(\alpha_1|\boldsymbol{x}) - R(\alpha_2|\boldsymbol{x})$, in order to minimize the expected loss, we have the following decision rule:

$$\begin{cases} choose\ C_1,\ if\ g(\boldsymbol{x}) < 0 \\ choose\ C_2,\ elsewhere \end{cases}$$

Then the decision boundary is the solution of $g(\boldsymbol{x}) = 0$.

$$\begin{aligned}
& g(\boldsymbol{x}) = 0 \\
& \Downarrow \\
& (\lambda_{11} - \lambda_{21}) \frac{P(\boldsymbol{x}|C_1)P(C_1)}{P(\boldsymbol{x})} \\
& = (\lambda_{22} - \lambda_{12}) \frac{P(\boldsymbol{x}|C_2)P(C_2)}{P(\boldsymbol{x})} \\
& \Downarrow \\
& \log(|\lambda_{11} - \lambda_{21}|) + \log P(\boldsymbol{x}|C_1) + \log P(C_1) \\
& = \log(|\lambda_{22} - \lambda_{12}|) + \log P(\boldsymbol{x}|C_2) + \log P(C_2) \\
& \Downarrow \\
& \log(|\lambda_{11} - \lambda_{21}|) + \frac{1}{2\sigma^2}(\boldsymbol{2\mu_1^T x - \mu_1^T \mu_1}) + \log P(C_1) \\
& = \log(|\lambda_{22} - \lambda_{12}|) + \frac{1}{2\sigma^2}(\boldsymbol{2\mu_2^T x - \mu_2^T \mu_2}) + \log P(C_2) \\
& \Downarrow
\end{aligned}$$

$$\frac{1}{\sigma^2}(\boldsymbol{\mu_1^T - \mu_2^T})\boldsymbol{x} = \log \frac{(\lambda_{22} - \lambda_{12})P(C_2)}{(\lambda_{11} - \lambda_{21})P(C_1)} + \frac{1}{2\sigma^2}(\boldsymbol{\mu_1^T \mu_1 - \mu_2^T \mu_2})$$

Define $\boldsymbol{w} = \frac{1}{\sigma^2}(\boldsymbol{\mu_1 - \mu_2})$ and $w_0 = \log \frac{(\lambda_{11}-\lambda_{21})P(C_1)}{(\lambda_{22}-\lambda_{12})P(C_2)} - \frac{1}{2\sigma^2}(\boldsymbol{\mu_1^T \mu_1 - \mu_2^T \mu_2})$, we have

$$\boldsymbol{w^T x} + w_0 = \boldsymbol{w^T (x - x_0)} = 0$$

where $\boldsymbol{x_0} = \frac{1}{2}(\boldsymbol{\mu_1 + \mu_2}) - \frac{1}{||\boldsymbol{\mu_1 - \mu_2}||_2^2} \sigma^2 \log \frac{(\lambda_{11}-\lambda_{21})P(C_1)}{(\lambda_{22}-\lambda_{12})P(C_2)} (\boldsymbol{\mu_1 - \mu_2})$

However, in order to minimize the probability of error, we define

$$\begin{aligned}
g'(\boldsymbol{x}) &= \log P(C_1|\boldsymbol{x}) - \log P(C_2|\boldsymbol{x}) \\
&= \log P(\boldsymbol{x}|C_1) - \log P(\boldsymbol{x}|C_2) + \log \frac{P(C_1)}{P(C_2)} \\
&= \frac{1}{\sigma^2}(\boldsymbol{\mu_1^T} - \boldsymbol{\mu_2^T})\boldsymbol{x} - \frac{1}{2\sigma^2}(\boldsymbol{\mu_1^T \mu_1} - \boldsymbol{\mu_2^T \mu_2}) + \log \frac{P(C_1)}{P(C_2)} \\
&= \boldsymbol{w'^T}\boldsymbol{x} + w_0{}' = \boldsymbol{w'^T(x - x_0{}')} = 0 \\
\Rightarrow \boldsymbol{x_0{}'} &= \frac{1}{2}(\boldsymbol{\mu_1} + \boldsymbol{\mu_2}) - \frac{1}{||\boldsymbol{\mu_1} - \boldsymbol{\mu_2}||_2^2}\sigma^2 \log \frac{P(C_1)}{P(C_2)}(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})
\end{aligned}$$

The corresponding decision rule is:

$$\begin{cases} choose\ C_1,\ if\ g'(\boldsymbol{x}) > 0 \\ choose\ C_2,\ elsewhere \end{cases}$$

Based on that, we know

$$\begin{aligned}
&\boldsymbol{w} = \boldsymbol{w'} \\
&x_0 = x_0{}' - \frac{1}{||\boldsymbol{\mu_1} - \boldsymbol{\mu_2}||_2}\sigma^2 \log \frac{\lambda_{11} - \lambda_{21}}{\lambda_{22} - \lambda_{12}}(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})
\end{aligned}$$

Therefore, two decision boundaries have the same normal vector but different bias.

(b) 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 $P(C_1) = P(C_2)$.

Solution:

By Bayesian rule, we have

$$P(C_i|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|C_i)P(C_i)}{P(\boldsymbol{x})}$$

Given that x_1, x_2 are independent following the Laplace distribution, we have

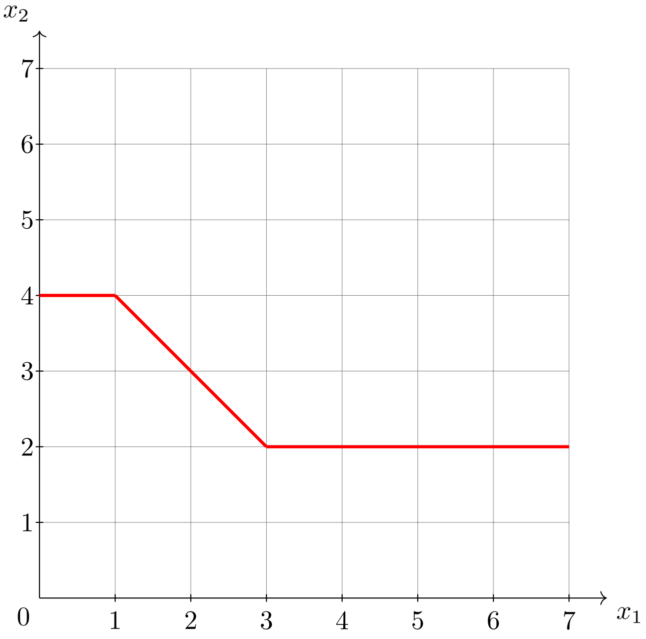
$$P(C_i|\boldsymbol{x}) = \frac{P(x_1|C_i)P(x_2|C_i)P(C_i)}{P(\boldsymbol{x})}$$

Then define discrimination function

$$\begin{aligned}
g_i(\boldsymbol{x}) &= \log P(C_i|\boldsymbol{x}) \\
&= \log P(x_1|C_i) + \log P(x_2|C_i) + \log P(C_i) - \log P(\boldsymbol{x}) \\
&= -2\log 2\sigma - \frac{1}{\sigma}|x_1 - \mu_{i1}| - \frac{1}{\sigma}|x_2 - \mu_{i2}| + \log P(C_i) \\
g(\boldsymbol{x}) &= g_1(\boldsymbol{x}) - g_2(\boldsymbol{x}) \\
&= -|x_1 - 1| - |x_2 - 1| + |x_1 - 3| + |x_2 - 5|
\end{aligned}$$

Then let $g(\boldsymbol{x} = 0)$, we get:

$$\begin{aligned}
&-|x_1 - 1| - |x_2 - 1| + |x_1 - 3| + |x_2 - 5| = 0 \\
&\Rightarrow x_2 = \begin{cases} 4, & for\ x_1 < 1 \\ -x_1 + 5 = 0, & for\ 1 \leq x_1 \geq 3 \\ 2, & for\ x_1 > 3 \end{cases}
\end{aligned}$$


x_2
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
x_1

HW2-Coding

October 24, 2022

1 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.

1.1 Part 0: Preparation before training.

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

```
[1]: #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
```

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

```
[3]: # Size of features and labels
     data.shape, target.shape
```

```
[3]: ((569, 30), (569,))
```

```
[4]: #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)

```
[5]: # 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 comment below.
     """
     scaler = preprocessing.StandardScaler().fit(X_train)
     X_train = scaler.transform(X_train)
     X_test = scaler.transform(X_test)
     """
     #Your codes below:
     mean = np.mean(X_train, axis=0)
     std = np.std(X_train, axis=0)
     X_train = (X_train - mean) / std
     X_test = (X_test - mean) / std
```

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

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

```
[7]: #Function to calculate TPR,FPR,TNR and FNR to be included in confusion matrix
     def find_rates(mat):
         mat2 = [(mat[0, 0]), (mat[1, 0]), (mat[0, 1]), (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
```

1.2 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.Logis

```
[8]: #Logistic regression using sklearn
     LRexample = LogisticRegression(penalty='l2', C=0.1, solver='liblinear')
```

```
       LRexample.fit(X_train, y_train)
```

```
 [8]: LogisticRegression(C=0.1, solver='liblinear')
```

```
 [9]: # Predict on the test set
      y_pred_sklearn = LRexample.predict(X_test)
```

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

```
[10]: (array([0, 1]), array([ 67, 121]))
```

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

```
[11]: (array([0, 1]), array([ 67, 121]))
```

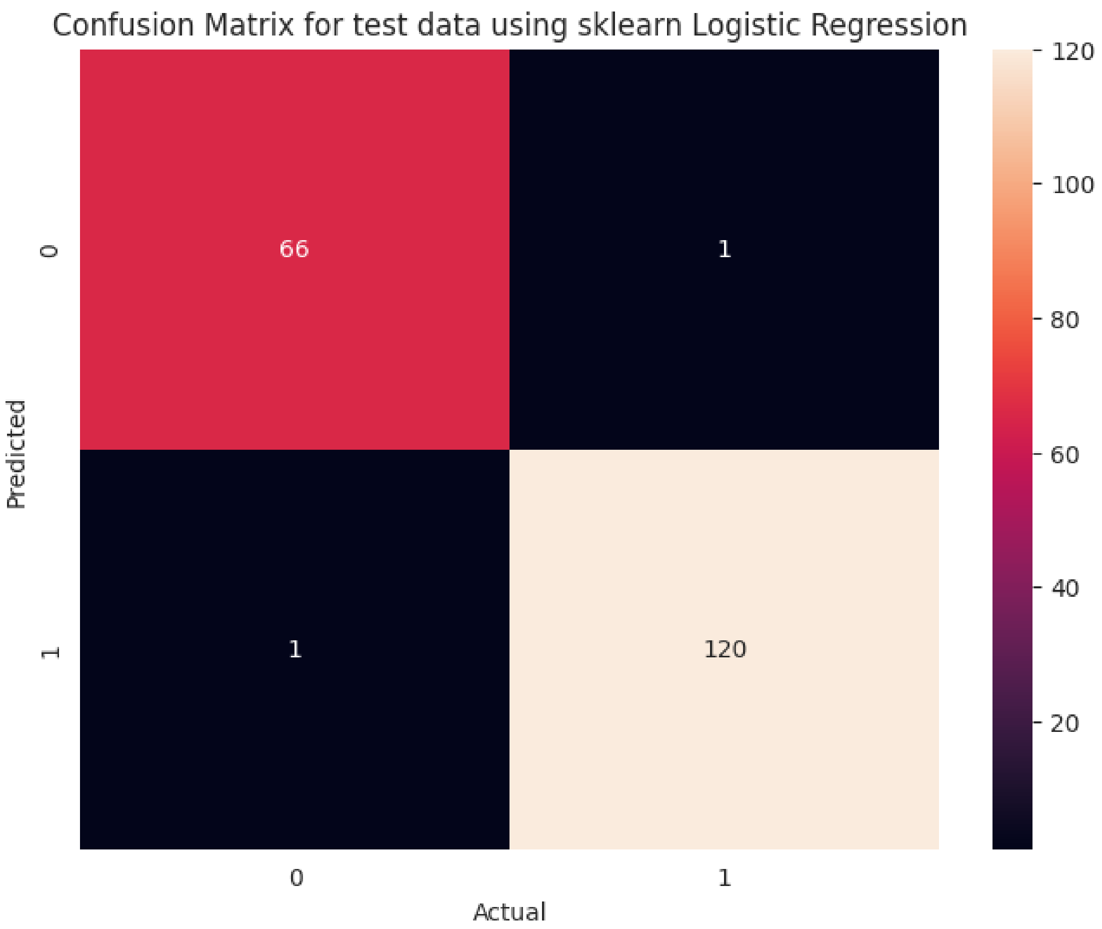
```
[12]: # true-negative, false-negative, false-negative, true-positive
      tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
      (tn, fp, fn, tp)
```

```
[12]: (66, 1, 1, 120)
```

```
[13]: 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')
```

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



[14]:
```
LRexample.score(X_test, y_test)
coef = LRexample.coef_[0].copy()
```

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

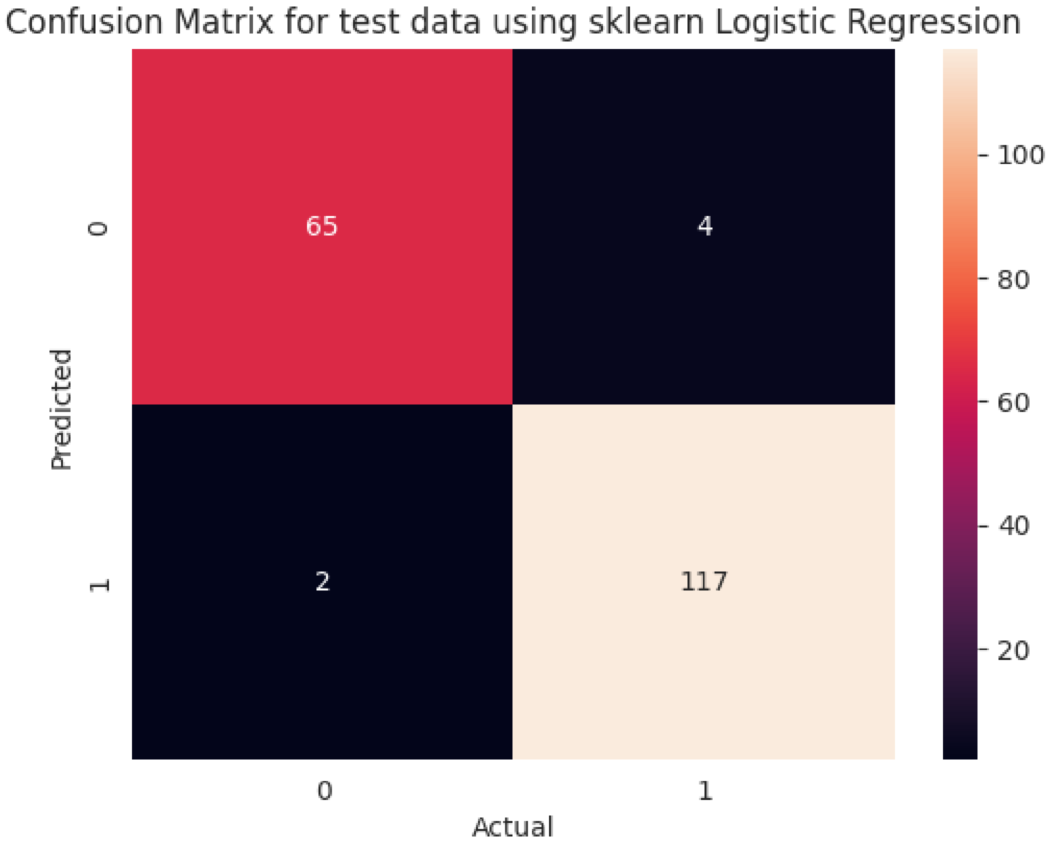
penalty='l2' means that we use l2 norm of all parameters $||\mathbf{w}||_2^2$ to do regularization. C=0.1 means we set the regularization strength λ to $\frac{1}{C} = 10$ so that the the smaller C is, the larger the penalty is. solver='liblinear' means that we use coordinate descent algorithm to optimize the model.

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

[15]:
```
LRexample = LogisticRegression(penalty='l1', C=0.1, solver='liblinear')
LRexample.fit(X_train, y_train)
# Predict on the test set
y_pred_sklearn = LRexample.predict(X_test)
# The labels of ground-truth on test set.
np.unique(y_test, return_counts=True)
# The labels produced by LR model on test set.
```

```python
np.unique(y_pred_sklearn, return_counts=True)
# true-negative, false-negative, false-negative, true-positive
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
print(f"tn={tn},fp={fp},fn={fn},tp={tp}")
mat_test = find_rates(confusion_matrix(y_test, y_pred_sklearn))
plt.title('Confusion Matrix for test data using sklearn Logistic Regression')
sns.heatmap(mat_test, annot=True, fmt='g')
print(f"score={LRexample.score(X_test, y_test)}")
```

```
tn=65,fp=2,fn=4,tp=117
score=0.9680851063829787
```

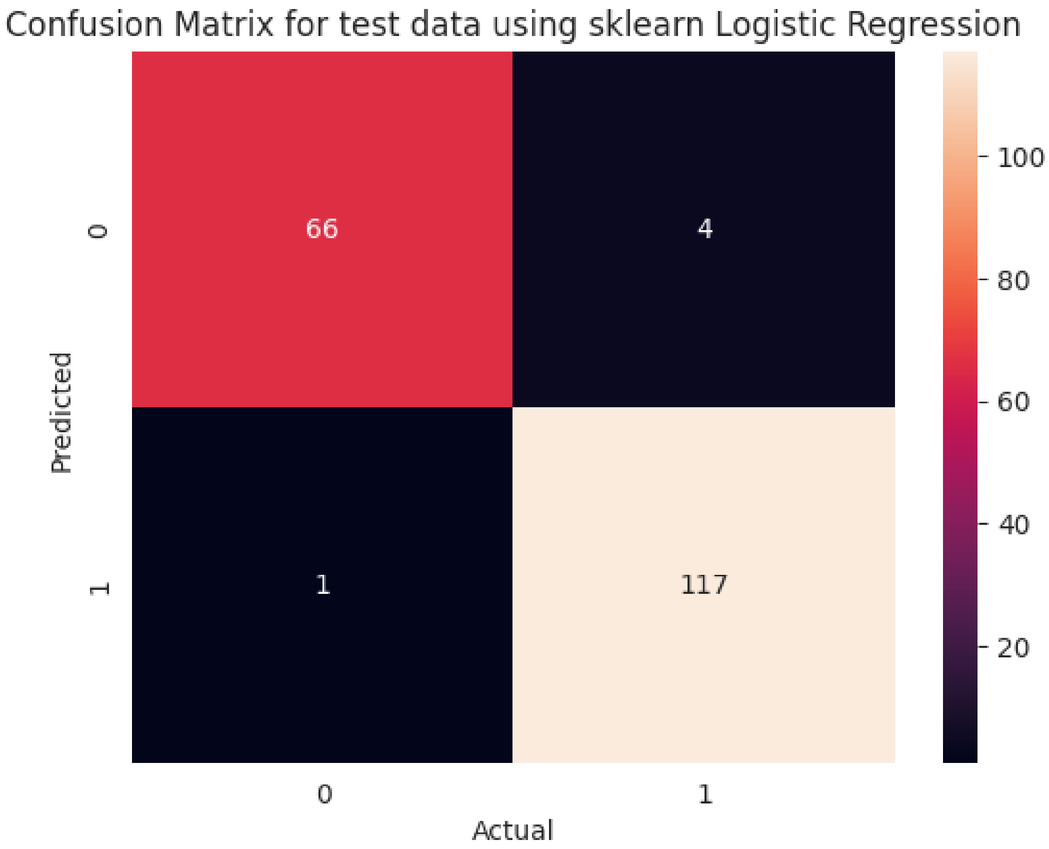


[16]:

```python
LRexample = LogisticRegression(penalty='l1', C=1.0, solver='liblinear')
LRexample.fit(X_train, y_train)
# Predict on the test set
y_pred_sklearn = LRexample.predict(X_test)
# The labels of ground-truth on test set.
np.unique(y_test, return_counts=True)
# The labels produced by LR model on test set.
np.unique(y_pred_sklearn, return_counts=True)
```

```
# true-negative, false-negative, false-negative, true-positive
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
print(f"tn={tn},fp={fp},fn={fn},tp={tp}")
mat_test = find_rates(confusion_matrix(y_test, y_pred_sklearn))
plt.title('Confusion Matrix for test data using sklearn Logistic Regression')
sns.heatmap(mat_test, annot=True, fmt='g')
print(f"score={LRexample.score(X_test, y_test)}")
```

```
tn=66,fp=1,fn=4,tp=117
score=0.973404255319149
```

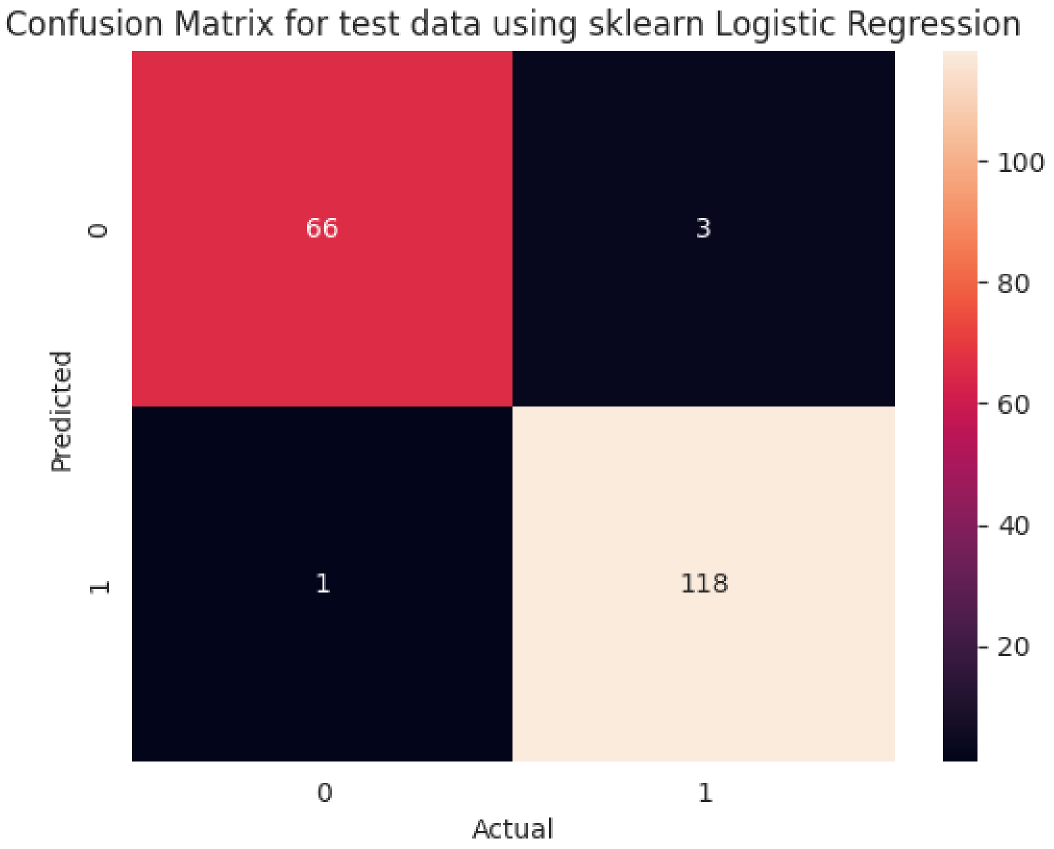


[17]:

```
LRexample = LogisticRegression(penalty='l2', C=1.0, solver='liblinear')
LRexample.fit(X_train, y_train)
# Predict on the test set
y_pred_sklearn = LRexample.predict(X_test)
# The labels of ground-truth on test set.
np.unique(y_test, return_counts=True)
# The labels produced by LR model on test set.
np.unique(y_pred_sklearn, return_counts=True)
# true-negative, false-negative, false-negative, true-positive
```

```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
print(f"tn={tn},fp={fp},fn={fn},tp={tp}")
mat_test = find_rates(confusion_matrix(y_test, y_pred_sklearn))
plt.title('Confusion Matrix for test data using sklearn Logistic Regression')
sns.heatmap(mat_test, annot=True, fmt='g')
print(f"score={LRexample.score(X_test, y_test)}")
```

```
tn=66,fp=1,fn=3,tp=118
score=0.9787234042553191
```

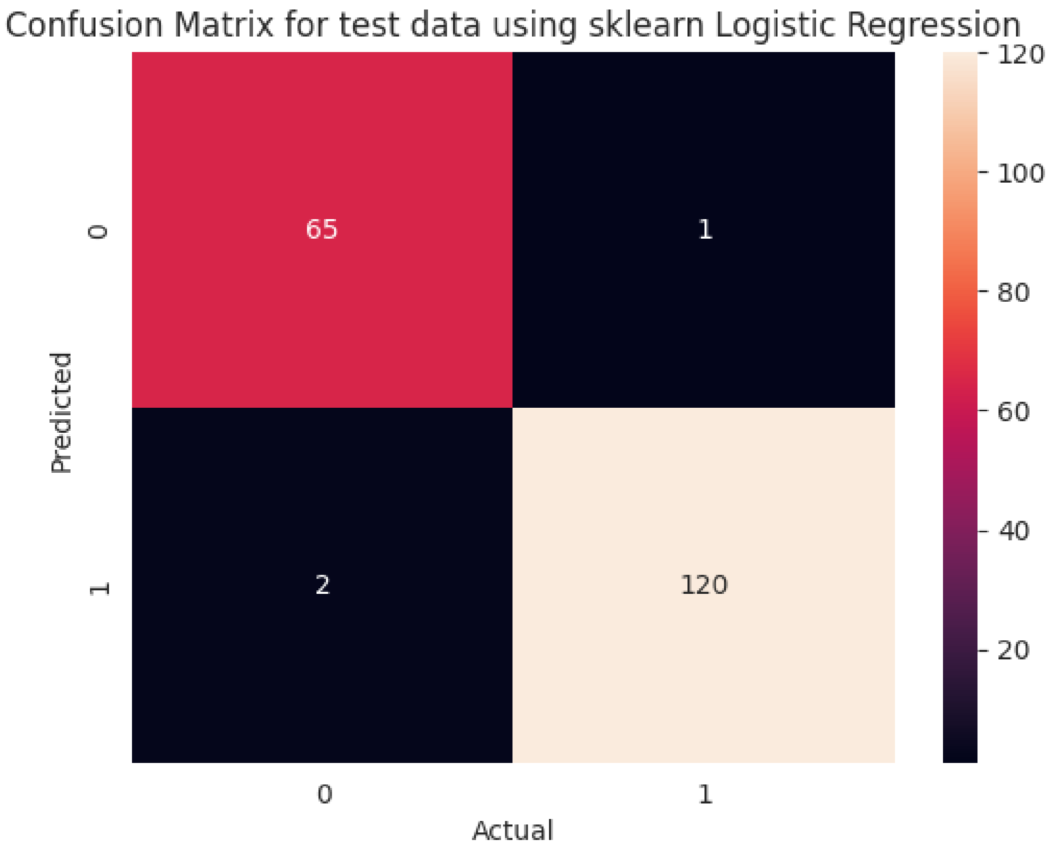


[18]:

```
LRexample = LogisticRegression(penalty='l2', C=0.1, solver='newton-cg')
LRexample.fit(X_train, y_train)
# Predict on the test set
y_pred_sklearn = LRexample.predict(X_test)
# The labels of ground-truth on test set.
np.unique(y_test, return_counts=True)
# The labels produced by LR model on test set.
np.unique(y_pred_sklearn, return_counts=True)
# true-negative, false-negative, false-negative, true-positive
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_sklearn).ravel()
```

```
print(f"tn={tn},fp={fp},fn={fn},tp={tp}")
mat_test = find_rates(confusion_matrix(y_test, y_pred_sklearn))
plt.title('Confusion Matrix for test data using sklearn Logistic Regression')
sns.heatmap(mat_test, annot=True, fmt='g')
print(f"score={LRexample.score(X_test, y_test)}")
```

```
tn=65,fp=2,fn=1,tp=120
score=0.9840425531914894
```



1.3 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.

1.3.1 Part 2.1: Implement logistic regression using Batch-GD

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

Batch-Gradient-Descent algorithm means all training data are visible. For logistic regression problem with 2 classes, we assume each class follows the Bernoulli distribution. Then the likelihood is:

$$L(\mathbf{w}|\mathcal{X}) = \prod_{t=1}^{N} (y^t)^{r^t} (1 - y^t)^{1-r^t}$$

And the regularized loss function is:

$$E(\mathbf{w}|\mathcal{X}) = -\log L(\mathbf{w}|\mathcal{X}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 = -\sum_t \left[r^t \log y^t + (1 - r^t)\log(1 - y^t)\right] + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

where $y_i^t = sigmoid(\mathbf{wx})$. Then the corresponding update rules for $\mathbf{w}$ and w_0 are as following:

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t (r_j^t - y_j^t) x_j^t - \eta \lambda w_j, \quad for\ j = 1, 2 \cdots, d$$

$$\Delta w_0 = -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^t - y^t) - \eta \lambda w_0$$

$$w_j = w_j + \Delta w_j$$

$$w_0 = w_0 + \Delta w_0$$

[19]:

```
"""
At each iteration, train all the samples and update weights. The initialization␣
↪point should be set to all-zero vector.
"""
n_iter = 50  # number of iterations
reg = 0.01  # regularization parameter lambda
r = 0.1  # learning rate
N, d = X_train.shape
w_BGD = np.zeros((d + 1))
for j in range(n_iter):
    delta = np.zeros((d + 1))
    for it in range(N):
        input = np.insert(X_train[it], 0, 1, axis=0)
        y = 1 / (1 + np.exp(-np.dot(w_BGD, input)))
        delta += (y_train[it] - y) * input
    w_BGD += r * (delta - reg * w_BGD)
```

```
/tmp/ipykernel_7358/1462154002.py:13: RuntimeWarning: overflow encountered in
exp
  y = 1 / (1 + np.exp(-np.dot(w_BGD, input)))
```

[20]:

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

[21]:

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

[21]:

```
(array([0, 1]), array([ 67, 121]))
```

[22]:

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

[22]:

```
(array([0, 1]), array([ 69, 119]))
```

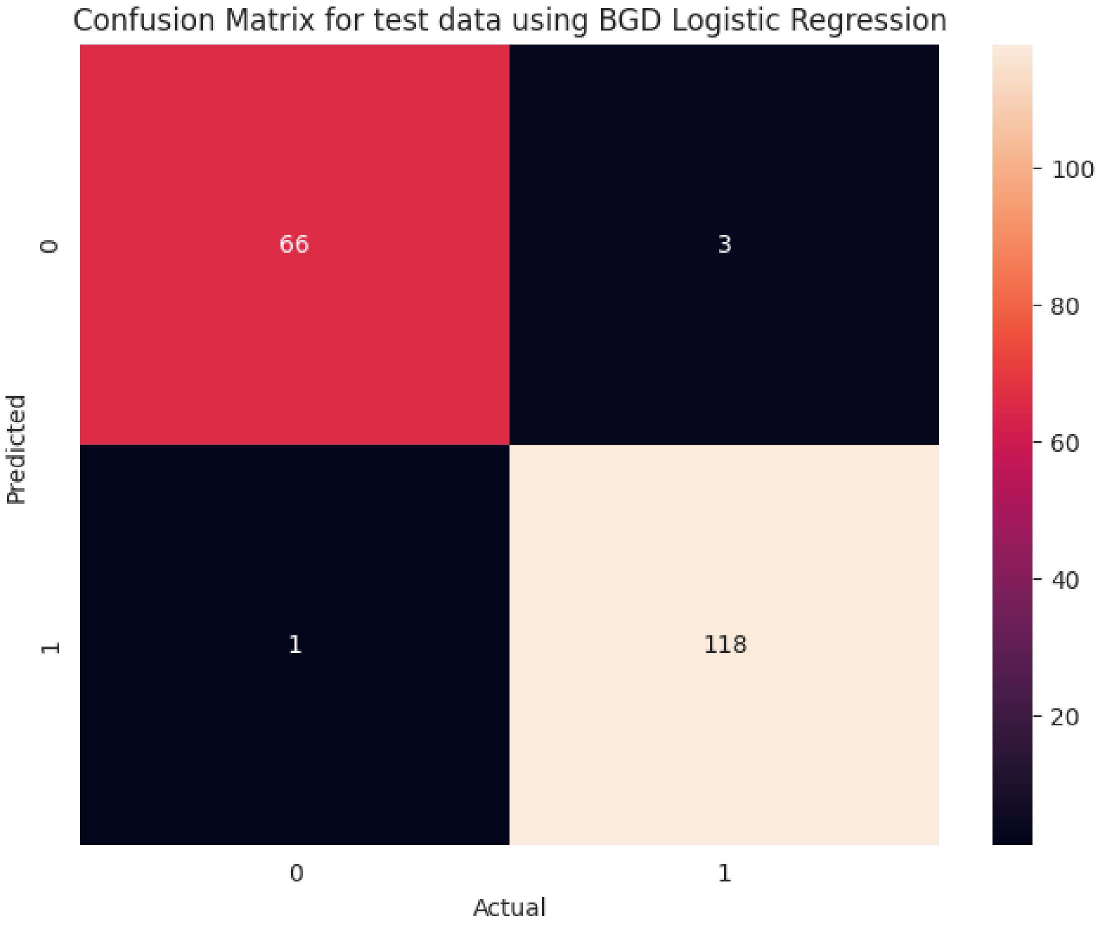
[23]:

```python
# 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')
```

[23]:

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



1.3.2 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 for more details with respect to SGD, momentum and more acceleration tricks.

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

SGD with momentum algorithm means only part of training data are visible and we use momentum to accelerate convergence. For logistic regression problem with 2 classes, we assume each class follows the Bernoulli distribution. Then the likelihood is:

$$L(\mathbf{w}|\mathcal{X}) = \prod_{t=1}^{M} (y^t)^{r^t}(1-y^t)^{1-r^t}$$

And the regularized loss function is:

$$E(\mathbf{w}|\mathcal{X}) = -\frac{1}{M}\log L(\mathbf{w}, w_0|\mathcal{X}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 = -\frac{1}{M}\sum_t \left[r^t \log y^t + (1-r^t)\log(1-y^t)\right] + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

where $y_i^t = sigmoid(\mathbf{wx})$. Then the corresponding update rules for velocity v and parameter $\mathbf{w}$ are as following:

$$\Delta w_j = \alpha\Delta w_j - \eta\frac{\partial E}{\partial w_j} = \alpha\Delta w_j + \eta\frac{1}{M}\sum_t (r_j^t - y_j^t)x_j^t - \eta\lambda w_j, \quad for\ j = 1, 2\cdots, d$$

$$\Delta w_0 = \alpha\Delta w_0 - \eta\frac{\partial E}{\partial w_0} = \alpha\Delta w_0 + \eta\frac{1}{M}\sum_t (r^t - y^t) - \eta\lambda w_0$$

$$w_j = w_j + \Delta w_j$$

$$w_0 = w_0 + \Delta w_0$$

[24]:
```
"""
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␣
 ↪initialization point should be set to all-zero vector.

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, d = X_train.shape
w_SGD = np.zeros((d + 1))
v = 0
for j in range(n_iter):
    np.random.seed(j)
    idx = np.random.randint(X_train.shape[0], size=sample_size)
    # Do NOT modify codes above, especially the random code.
    # At each iteration, choose samples from X_train, y_train, with index idx.
    # Your codes below:
    delta = 0
    for i in idx:
        input = np.insert(X_train[i], 0, 1, axis=0)
        y = 1 / (1 + np.exp(-np.dot(w_SGD, input)))
        delta += r * (y_train[i] - y) * input
    delta /= sample_size
    delta -= r * reg * w_SGD
    delta += momen * v
    v = delta
    w_SGD += delta
```

[25]:
```
#Getting predictions for test datapoints
y_pred_SGD = predict(X_test, w_SGD)
```

[26]:
```
np.unique(y_test, return_counts=True)
```

[26]:
```
(array([0, 1]), array([ 67, 121]))
```

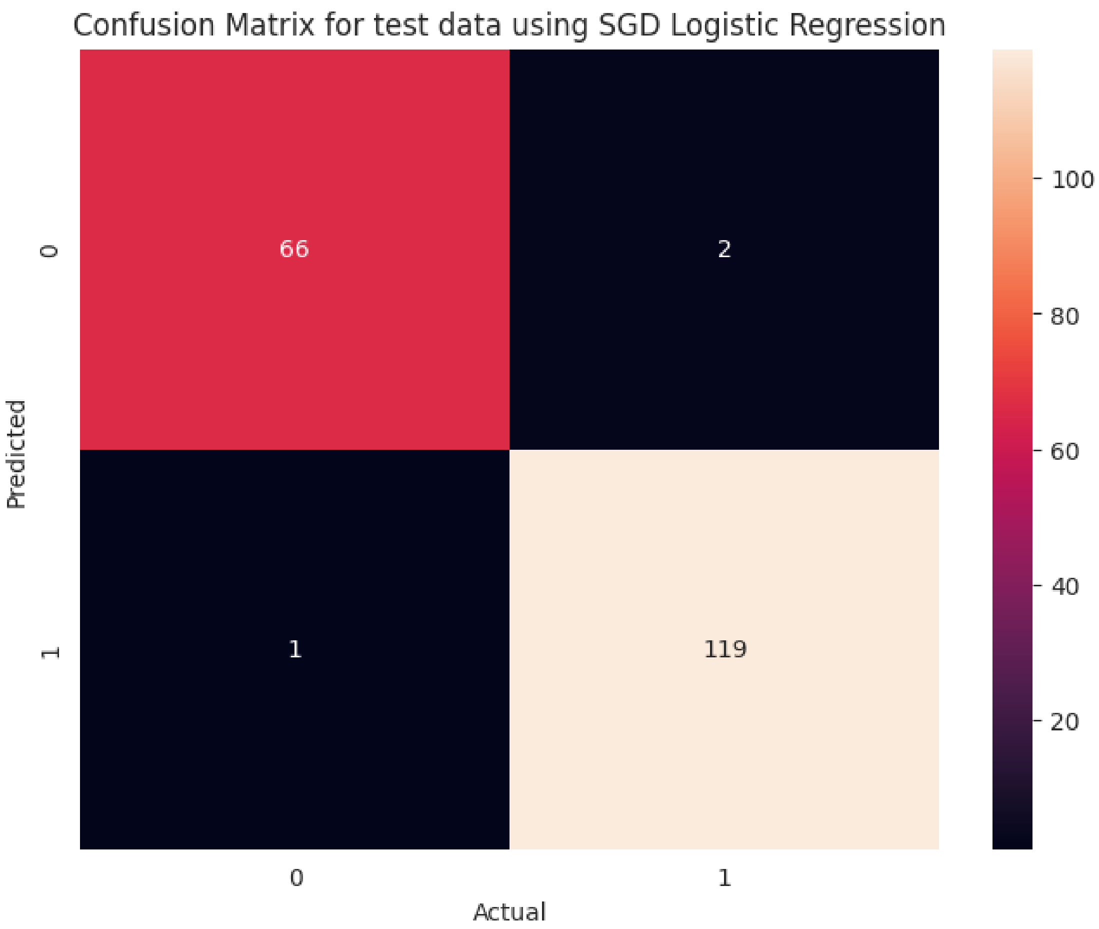
[27]:
```
np.unique(y_pred_SGD, return_counts=True)
```

[27]:
```
(array([0, 1]), array([ 68, 120]))
```

[28]:
```
# 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')
```

[28]:
```
<AxesSubplot:title={'center':'Confusion Matrix for test data using SGD Logistic
Regression'}, xlabel='Actual', ylabel='Predicted'>
```



[29]:

```
# Print a table to show every coefficient in vector w, and compute the absolute␣
 ↪difference between coefficients of BGD and SGD with momentum methods.

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(1, 31):
    p.add_row(['{:.5f}'.format(coef[i - 1]), '{:.5f}'.format(w_BGD[i]),
               '{:.5f}'.format(w_SGD[i]), '{:.5f}'.format(abs(w_BGD[i] -␣
 ↪w_SGD[i]))])
print(p)
# LRclf.coef_[0, i]
```

```
+----------------------------------------------+
|          Weights from both models            |
+----------+-----------+----------+------------+
| SKlearn  |    BGD    |   SGD    | Difference |
+----------+-----------+----------+------------+
| -0.33269 |  -3.60382 | -0.36479 |  3.23902   |
| -0.35660 |  -5.84576 | -0.34029 |  5.50546   |
| -0.32618 |  -3.50836 | -0.36234 |  3.14603   |
| -0.33995 |  -4.43808 | -0.35129 |  4.08679   |
| -0.12556 |  -4.50551 | -0.17064 |  4.33486   |
| 0.03085  |  2.92869  | -0.11668 |  3.04537   |
| -0.37123 |  -6.00063 | -0.28407 |  5.71656   |
| -0.47501 |  -6.96736 | -0.40054 |  6.56681   |
| -0.04295 |  -0.36137 | -0.07933 |  0.28204   |
| 0.18929  |  2.51731  | 0.22256  |  2.29475   |
| -0.45881 | -10.07003 | -0.31717 |  9.75285   |
| 0.02783  |  1.93975  | 0.01297  |  1.92678   |
| -0.35493 |  -7.35442 | -0.27850 |  7.07591   |
| -0.35434 |  -8.03878 | -0.28779 |  7.75099   |
| -0.07387 |  -0.83634 | -0.03772 |  0.79862   |
| 0.19670  |  6.49909  | 0.05322  |  6.44587   |
| 0.04585  |  0.95291  | 0.10556  |  0.84735   |
| -0.10702 |  -2.11120 | -0.06499 |  2.04621   |
| 0.13595  |  4.51450  | 0.14164  |  4.37286   |
| 0.24279  |  6.94723  | 0.22830  |  6.71893   |
| -0.45043 |  -8.17931 | -0.42200 |  7.75732   |
| -0.53362 | -11.34376 | -0.42549 |  10.91827  |
| -0.41601 |  -7.27211 | -0.41231 |  6.85980   |
| -0.42044 |  -8.15857 | -0.39113 |  7.76744   |
| -0.34684 |  -8.10016 | -0.32901 |  7.77115   |
| -0.15248 |  0.20420  | -0.23539 |  0.43959   |
| -0.41778 |  -8.74473 | -0.33075 |  8.41398   |
| -0.44225 |  -8.95198 | -0.40644 |  8.54554   |
| -0.44613 |  -7.86921 | -0.25067 |  7.61853   |
| -0.08564 |  2.38279  | -0.07064 |  2.45344   |
+----------+-----------+----------+------------+
```