A.

```
X0 = X[t == 0]
X1 = X[t == 1]

m0 = X0.mean(axis=0)
m1 = X1.mean(axis=0)


cov_inclass =np.matmul(np.transpose((X0-m0)),(X0-m0))+np.matmul(np.transpose((X1-m1)),(X1-m1))
self.w = np.matmul(np.linalg.inv(cov_inclass),m1-m0)


self.w /= np.linalg.norm(self.w).clip(min=1e-10)
self._threshold(X, t)
```

B.

```
for _ in range(max_iter):
    w_prev = np.copy(w)
    y_hat = self.proba(X)
    grad = np.matmul((y_hat-t),X)
    w = gradient_descent(w, grad, learning_rate=0.1)
    if np.allclose(w, w_prev):
        break
```

C.

Fisher LDA uses assumption that the data set come from Gaussian normal distribution.
Also Fisher LDA uses projection concepts. The bold type **m1, m2** are the average of data and normal type m1, m2 are projection to the 1 dimension of **m1, m2.**

$$\mathbf{m1} = \frac{1}{N}\sum_{n\in C1} x_n$$
$$\mathbf{m2} = \frac{1}{N}\sum_{n\in C2} x_n$$

$$\mathrm{m1} = w^T\boldsymbol{m1}$$
$$\mathrm{m2} = w^T\boldsymbol{m2}$$
$$s_1{}^2 = \sum_{n\in C1}(y_n - m1)^2 = \sum_{n\in C1}(w^T x - m1)^2$$

$$s_2{}^2 = \sum_{n \in C2} (y_n - m2)^2 = \sum_{n \in C2} (w^T x - m2)^2$$

$$s_1{}^2 + s_2{}^2 \ \ is \ with - in \ \ class \ \ variance$$
$$(\mathrm{m2} - \mathrm{m1})^2 \ \ is \ \ between \ \ class \ \ variance$$

Our purpose is to maximize the ratio of Sb(between class variance) per Sw(within class variance) which means making small Sw and big Sb. This is what we call Fisher criterion

$$\mathrm{J(w)} = \frac{(\mathrm{m2} - \mathrm{m1})^2}{s_1{}^2 + s_2{}^2}$$

After algebraic calculation, we can get this equation.

$$\mathrm{J(w)} = \frac{w^T S_B w}{w^T S_w w}$$

Where,

$$S_B = (\boldsymbol{m2} - \boldsymbol{m1})(\mathbf{m2} - \mathbf{m1})^T$$

$$S_W = \sum_{n \in C1} (x_n - \boldsymbol{m1})(x_n - \mathbf{m1})^T + \sum_{n \in C2} (x_n - \boldsymbol{m2})(x_n - \mathbf{m2})^T$$

If we get derivative of J(w), it is maximized when this equation is satisfied.

$$(w^T S_B w) S_w w = (w^T S_w w) S_B w$$

Then we can get the weights by doing some algebra to this equation, and r is just constant which means scalar we have in this equation.

$$w = r\mathrm{S}^{-1}{}_w(\boldsymbol{m2} - \boldsymbol{m1})$$

D.

The assumptions of logistic regression : 1. Outcome is binary 2. Linear relationship between the logit of the outcome and each predictor variables 3. No high correlations between the predictors

The reason of probability function : Logistic regression is almost same as the linear regression. However the difference is that logistic regression uses logit function on the left side of regression equation like this; $\mathrm{logit}\big(\mathrm{p(C1|x)}\big) = w^T x$. From the definition of logit function we can rewrite the equation like this; $\mathrm{logit}\big(\mathrm{p(C1|x)}\big) = \log\left(\frac{\mathrm{p(C1|x)}}{1-\mathrm{p(C1|x)}}\right) = w^T x$. Then if we transform this equation, we can get the equation like this; $\mathrm{p(C1|x)} = \frac{1}{1+e^{-w^T x}}$. Since the definition of sigmoid function, we can represent the right side function as sigmoid function. We can use this as probability directly since

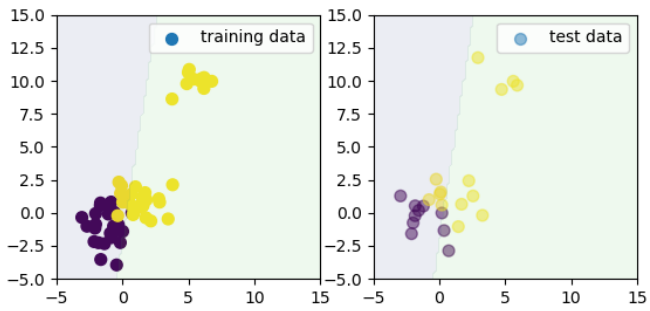
the sigmoid function is restricted from 0 to 1.

Loss function and gradient descent method : We are going to find the solution by gradient descent method. To use this method, the function that we want to apply should be convex function. That's why we use log to the original loss function and it finally comes to the cross entropy. (For more, this cross entropy can be divided into entropy and KL-divergence.)
We will make cost function as like this; $\mathrm{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if \ \ y = 1 \\ -\log(1-\hat{y}) & if \ \ y = 0 \end{cases}$
Then, loss function can be written as like this; $\mathrm{Loss(w)} = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1-\hat{y}^{(i)})]$
The gradient should be derivative of loss function by each weight; $\frac{\partial}{\partial \mathrm{w}_j} Loss(w) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})x^{(i)}$.

Then using gradient, we can iterate the gradient descent method by repeatedly alternate the old weight as new weight which is the (old weight-gradient*learning rate).

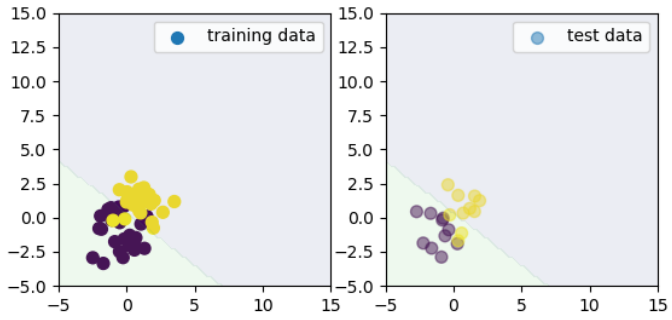
E.

For 5~9 data set, this result has come out.
[Average BCE]
2.8207
[BCE_std]
1.3023



For 0~4 data set, this result has come out.
[Average BCE]
1.9342

[BCE_std]
0.8219



Let me define the "sensitivity" as the change of slope of decision boundary. There is so much change in the slope of decision boundary between two cases. Also, Average BCE increases when there are outliers.

LDA is very sensitive to the outliers because we use the both variances which are within class variance and between class variance. LDA tends to find the weights that make small Sw and large Sb at the same time. That's why it is sensitive to the outliers.

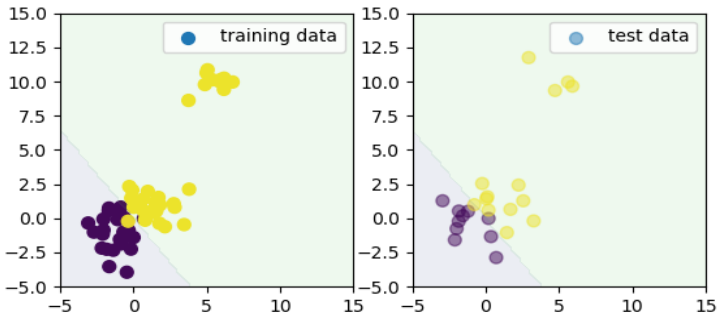
F.

***feature engineering = False**

When training and testing all together

[Average BCE]
1.7058
[BCE_std]
1.5526

For 5~9 data set, this result has come out.
[Average BCE]
1.4775
[BCE_std]
2.0103



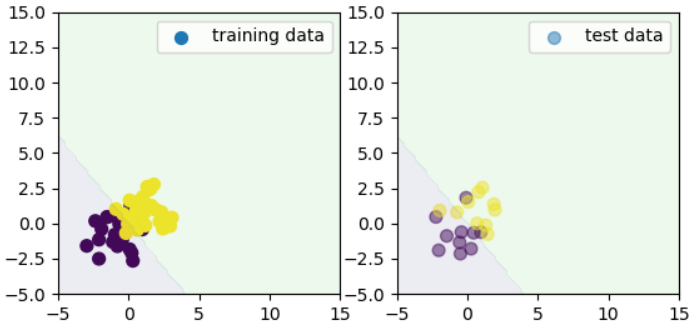
For 0~4 data set, this result has come out.

[Average BCE]

1.9342

[BCE_std]

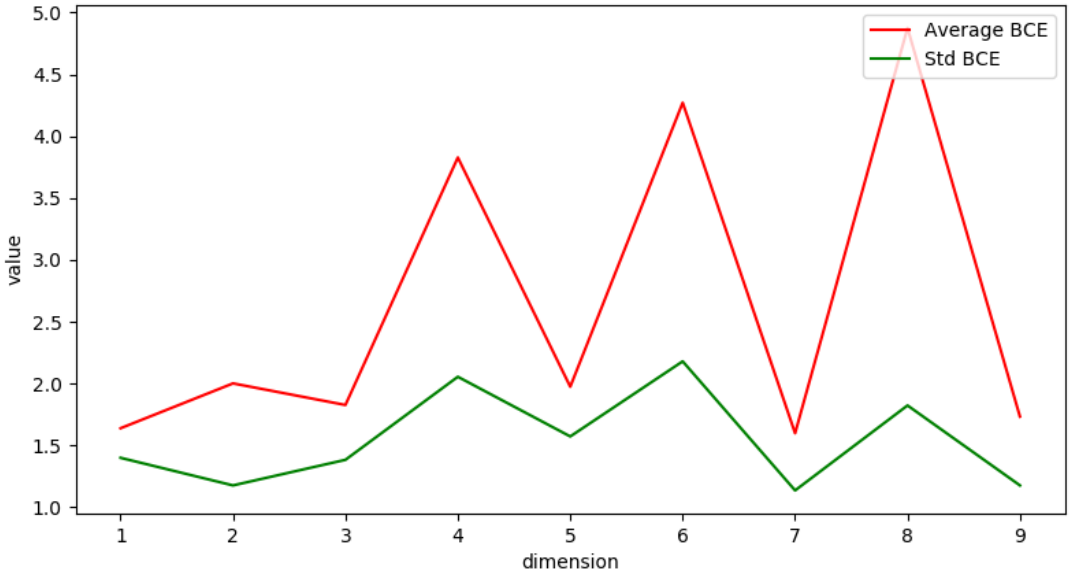
0.8219



Let me define the "sensitivity" as the change of slope of decision boundary. There is not much change in the slope of decision boundary between two cases. Also, Average BCE has small difference when there are outliers or not.

I think logistic regression is not sensitive to the outliers because it the outliers has extreme probability such as really close to 0 or 1, they rarely do not affect the decision boundary.

G.



I chose 1 dimension the average of BCE was 1.6387 and its std was 1.3995.
Dimension 7 also seems to have good average and standard deviation, however considering the complexity, lower complexity is better for our model.

When we use high even dimensions, logistic regression tends to classify the whole data set as Class 1, especially when there are outliers. This is because for even dimension there are many positive values of data after doing polynomial engineering. Therefore, sigmoid function classifies data point as class 1.