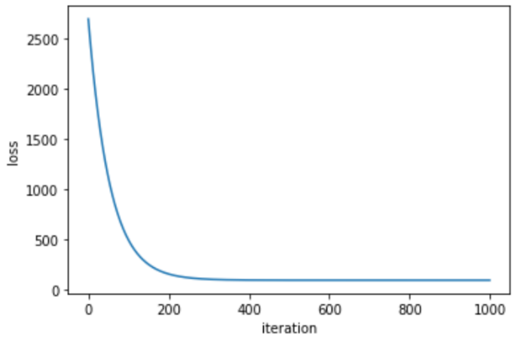
109550031_HW1

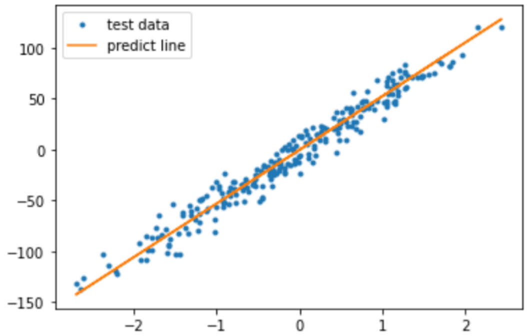
Part. 1, Coding

Linear regression model

1. Plot the learning curve of the training, you should find that loss decreases after a few iterations and finally converge to zero (x-axis=iteration, y-axis=loss, Matplotlib or other plot tools is available to use)



learning curve of the training data



testing data and prediction line

2. What's the Mean Square Error of your prediction and ground truth?

💨 Mean Square Error：110.42395623279842

```
def calculate_loss(self, y_pred, y_train):
    return (1/y_pred.size) * np.sum(np.square(y_pred - y_train))
```

```
x_test = np.reshape(x_test, x_test.size)
y_pred = regressor.predict(x_test)
MSE = regressor.calculate_loss(y_pred, y_test)
```

```
print("learning rate {}".format(learning_rate))
print("iteration     {}".format(iteration))
print("Mean Square Error:", MSE)
```

Code

```
learning rate 0.01
iteration    1000
Mean Square Error: 110.42395623279842
```

Result

3. What're the weights and intercepts of your linear model?

Weights	52.738914681987474
intercepts	-0.3344012407130547

```
def train(self, learning_rate, iteration, training_loss):
    self.learning_rate = learning_rate
    for it in range(iteration):
        y_pred = self.predict(self.x_train)
        loss = self.calculate_loss(y_pred, self.y_train)
        training_loss.append(loss)
        self.update_weight(y_pred)
    print(self.weight, self.intercept)
    return training_loss
```

```
regressor = Linear_Regression(x_train, y_train)
training_loss = regressor.train(learning_rate, iteration, training_loss)
```

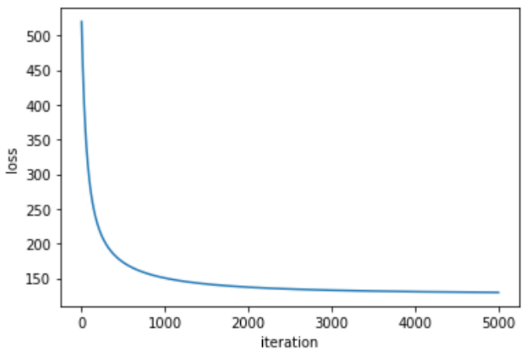
Code

```
52.738914681987474 -0.3344012407130547
```

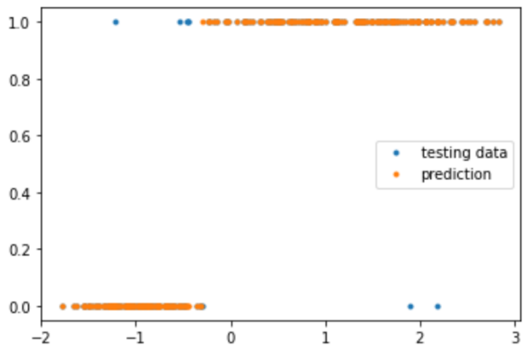
Result

Logistic regression model

1. Plot the learning curve of the training, you should find that loss decreases after a few iterations and finally converge to zero (x-axis=iteration, y-axis=loss, Matplotlib or other plot tools is available to use)



learning curve of the training data



testing data and prediction dot

2. What's the Cross Entropy Error of your prediction and ground truth?

Cross Entropy Error：45.3524718375564

```
def calculate_loss(self, H, y_train):
    len = H.size
    return -np.sum(y_train * np.log(H) + (1 - y_train) * np.log(1 - H))
```

```
H, y_pred = classifier.predict(np.reshape(x_test, x_test.size))
CEE = classifier.calculate_loss(H, y_test)
```

```
print("learning rate {}".format(learning_rate))
print("iteration     {}".format(iteration))
print("Cross Entropy Error:", CEE)
```

Code

```
learning rate 0.03
iteration     5000
Cross Entropy Error: 45.3524718375564
```

Result

3. What're the weights and intercepts of your linear model?

Weights	4.1853007504791515
intercepts	1.2927037660900267

```
def train(self, learning_rate, iteration, training_loss):
    self.learning_rate = learning_rate
    for it in range(iteration):
        H, y_pred = self.predict(self.x_train)
        H = np.reshape(H, H.size)
        loss = self.calculate_loss(H, self.y_train)
        training_loss.append(loss)
        self.update_weight(H)
    print(self.weight, self.intercept)
    return training_loss, self.weight, self.intercept
```

```
classifier = Linear_Classification(x_train, y_train)
training_loss, weight, intercept = classifier.train(learning_rate, iteration, training_loss)
```

Code

```
4.1853007504791515 1.2927037660900267
```

Result

Part. 2, Questions

1. What's the difference between Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent?

Difference 1：	how much data is taken into consideration in a single step
Gradient Descent	all the training data
Stochastic Gradient Descent	one data
Mini-Batch Gradient Descent	a batch of a fixed number of training data which is less than the actual dataset and call it a mini-batch

Difference 2：	use in different scenario
Gradient Descent	smaller datasets → moves directly towards an optimum solution
Stochastic Gradient Descent	larger datasets → the cost will fluctuate over the training examples and it will not necessarily decrease., but in the long run, the cost decreases with fluctuations
Mini-Batch Gradient Descent	fast computation → update parameters frequently and use vectorized implementation

2. Will different values of learning rate affect the convergence of optimization?

   Learning rate determines how fast or slow we will move towards the optimal weights. If learning rate is too small, we will need lots of iterations to converge to the best values. If the learning rate is very large, the optimal solution may be skipped. So using a good learning rate is crucial.

3. Show that the logistic sigmoid function (eq. 1) satisfies the property σ(−a) = 1 − σ(a) and that its inverse is given by $\sigma^{-1}(y)$ = ln {y/(1 − y)}.

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \qquad \text{(eq. 1)}$$

$$\boxed{\sigma(-a) = 1 - \sigma(a)}$$

<pf> $\sigma(-a) = \dfrac{1}{1+e^{a}}$

$= 1 - \dfrac{e^{a}}{1+e^{a}}$

$= 1 - \dfrac{1}{\frac{1}{e^{a}}+1}$

$= 1 - \dfrac{1}{1+e^{-a}}$

$= 1 - \sigma(a)$ #

$$\boxed{\sigma^{-1}(y) = \ln\left(\frac{y}{1-y}\right)}$$

<pf> $y = \sigma(a) = \dfrac{1}{1+e^{-a}}$

$\Rightarrow y + y \cdot e^{-a} = 1$

$\Rightarrow 1 + e^{-a} = \dfrac{1}{y}$

$\Rightarrow e^{-a} = \dfrac{1}{y} - 1 = \dfrac{1-y}{y}$

$\Rightarrow \ln(e^{-a}) = \ln\left(\dfrac{1-y}{y}\right)$

$\Rightarrow -a = \ln\left(\dfrac{1-y}{y}\right)$

$\Rightarrow a = \sigma^{-1}(y) = \ln\left(\dfrac{y}{1-y}\right)$ #

4. Show that the gradients of the cross-entropy error (eq. 2) are given by (eq. 3).

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk} \quad \text{(eq. 2)}$$

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\, \boldsymbol{\phi}_n \quad \text{(eq. 3)}$$

Hints:

$$a_k = \mathbf{w}_k^{\mathrm{T}} \boldsymbol{\phi}. \quad \text{(eq. 4)}$$

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \quad \text{(eq. 5)}$$

$$\frac{\partial E}{\partial y_{nk}} = -\frac{t_{nk}}{y_{nk}} \quad \text{——} \quad ①$$

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \quad \text{——} \quad ②$$

$$a_{nj} = w_j^T \phi_n \Rightarrow \nabla_{w_j} a_{nj} = \phi_n \quad \text{——} \quad ③$$

$$\frac{\partial E}{\partial a_{nj}} = \sum_{k=1}^{K} \frac{\partial E}{\partial y_{nk}} \cdot \frac{\partial y_{nk}}{\partial a_{nj}}$$

$$= -\sum_{k=1}^{K} \frac{t_{nk}}{y_{nk}} \cdot y_{nk}(I_{kj} - y_{nj})$$

$$= -\sum_{k=1}^{K} t_{nk}(I_{kj} - y_{nj})$$

$$= -t_{nj} + \sum_{k=1}^{K} t_{nk} \cdot y_{nj}$$

$$= y_{nj} - t_{nj} \quad \text{——} \quad ④$$

$$\left(\forall n, \sum_{k} t_{nk} = 1\right)$$

$$\nabla_{w_j} E(w_1, \dots, w_K) = \nabla_{w_j} a_{nj} \cdot \frac{\partial E}{\partial a_{nj}}$$

$$= \phi(n) \cdot \sum_{n=1}^{N} (y_{nj} - t_{nj})$$

$$= \sum_{n=1}^{N} (y_{nj} - t_{nj}) \cdot \phi(n) \quad \text{——} \quad (eq.3) \; \#$$