Logistic Regression

What is Logistic Regression?

- Logistic regression is a statistical method for predicting binary classes
  - Among a group of loan applicant, whether a person is good credit or bad?
  - Given an income level, whether a person will buy an iPhone or not

- The outcome or target variable is dichotomous in nature.

- Dichotomous means there are only two possible classes.

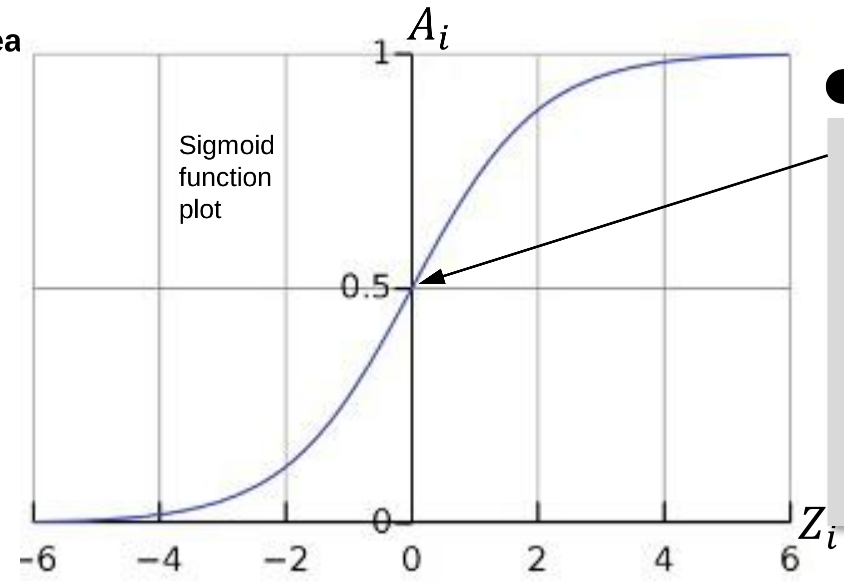
More Examples?

- Software project completion in time: Yes/No
- Marketing: Given a price point whether an item be sold
- Finance, banking: Will a stock gain? Should I give loan to the applicant
- Supply chain: Can meet delivery target or not
- Retail: Buyer / non buyer
- Property market: a house is easy to be sold or not

Logistic Regression

- Let's use a simple dataset with 100 houses (rows), 2 input features (columns) and 1 output to introduce and train our very first logistic regression model.
- Predict whether a house is easy to be sold or not based on the Size of the house and the number of Bedrooms.

Size (sq. ft.)	Bedrooms	Easy to sell or not
1600	5	1 (Yes)
1200	4	1 (Yes)
740	2	0 (No)
…	…	…
1091	4	0 (No)



0.5 is the default threshold.

If predicted A_i (i.e., probability) ≥ 0.5, rounded to 1.

If predicted A_i (i.e., probability) < 0.5, rounded to 0.

Logistic Regression

- Linear regression + Sigmoid function

$$Z_i = b + S_i \cdot w1 + B_i \cdot w2 \qquad A_i = \frac{1}{1 + e^{-Z_i}}$$

S_i is the Size of i^{th} house; B_i is the no. of Bedrooms of the i^{th} house
$w1$ and $w2$ are the weights; b is the bias/intercept

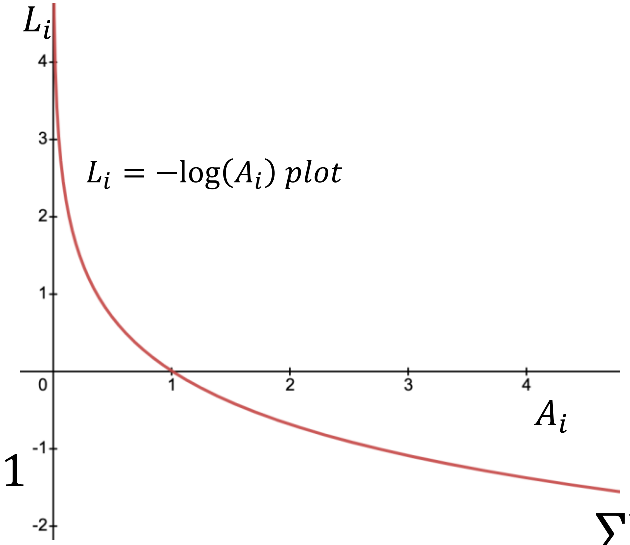
- Cross Entropy Loss Function

$$Loss_i = L_i = -[Y_i \log(A_i) + (1 - Y_i) \log(1 - A_i)]$$

When true class Y_i = 1,
$L_i = -\log(A_i)$

Want L_i close to 0,
then A_i should be close to 1

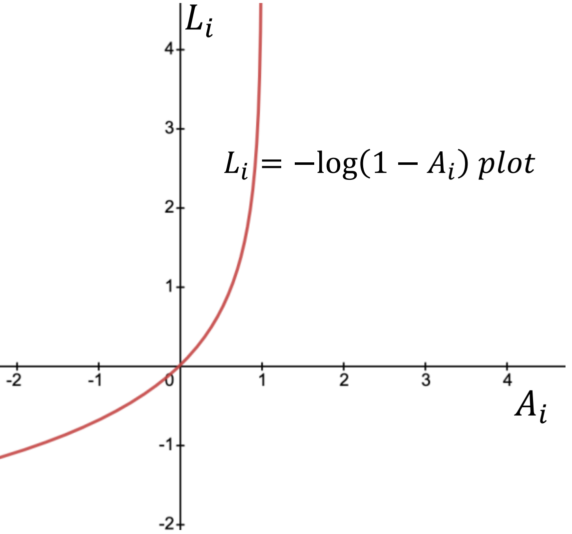
that is, the predicted A_i
should be close to true $Y_i = 1$



When true class Y_i = 0,
$L_i = -\log(1 - A_i)$

Want L_i close to 0,
Then A_i should be close to 0

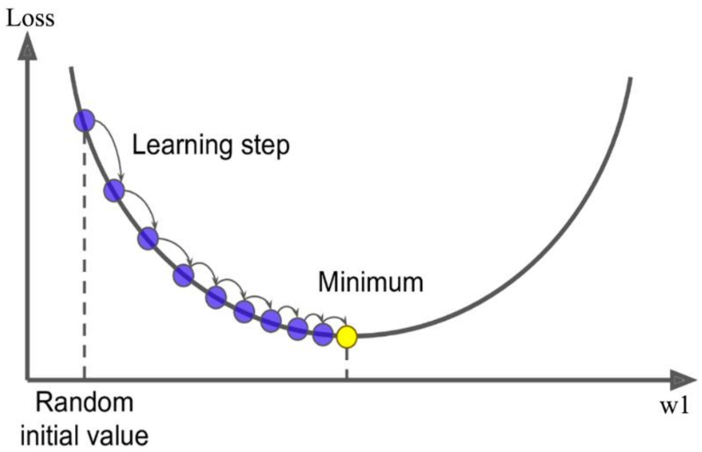
that is, the predicted A_i
should be close to true $Y_i = 0$



$$Mean\ Loss = L = \frac{\sum_{i=1}^{n} L_i}{n}$$

Sometimes, people also use J to denote mean loss.

Gradient Descent



$$\begin{cases} Z_i = b + S_i \cdot w1 + B_i \cdot w2 \\ A_i = \dfrac{1}{1+e^{-Z_i}} \\ L_i = -[Y_i \log(A_i) + (1-Y_i)\log(1-A_i)] \\ L = \dfrac{\sum_{i=1}^{n} L_i}{n} \end{cases}$$

$$\frac{\partial L_i}{\partial A_i} = -\frac{Y_i}{A_i} + \frac{1-Y_i}{1-A_i}$$

$$\frac{\partial L}{\partial w1} = \frac{\sum_{i=1}^{n}(\frac{\partial L_i}{\partial w1})}{n} = \frac{\sum_{i=1}^{n}(\frac{\partial L_i}{\partial A_i} \cdot \frac{\partial A_i}{\partial Z_i} \cdot \frac{\partial Z_i}{\partial w1})}{n} = \frac{\sum_{i=1}^{n}(A_i - Y_i) \cdot S_i}{n}$$

$$\frac{\partial A_i}{\partial Z_i} = A_i(1-A_i) \qquad \frac{\partial Z_i}{\partial w1} = S_i$$

Similarly,

$$\frac{\partial L}{\partial w2} = \frac{\sum_{i=1}^{n}(\frac{\partial L_i}{\partial A_i} \cdot \frac{\partial A_i}{\partial Z_i} \cdot \frac{\partial Z_i}{\partial w2})}{n} = \frac{\sum_{i=1}^{n}(A_i - Y_i) \cdot B_i}{n}$$

$$\frac{\partial L}{\partial b} = \frac{\sum_{i=1}^{n}(\frac{\partial L_i}{\partial A_i} \cdot \frac{\partial A_i}{\partial Z_i} \cdot \frac{\partial Z_i}{\partial b})}{n} = \frac{\sum_{i=1}^{n}(A_i - Y_i) \cdot 1}{n}$$

Then,

$$w1_{new} = w1_{old} - \frac{\partial L}{\partial w1} \cdot \alpha \qquad w2_{new} = w2_{old} - \frac{\partial L}{\partial w2} \cdot \alpha \qquad b_{new} = b_{old} - \frac{\partial L}{\partial b} \cdot \alpha$$

Logistic Regression Codes

$$\begin{cases} Z_i = b + S_i \cdot w1 + B_i \cdot w2 \\ A_i = \dfrac{1}{1+e^{-Z_i}} \\ L_i = -[Y_i \log(A_i) + (1-Y_i)\log(1-A_i)] \\ L = \dfrac{\sum_{i=1}^{n} L_i}{n} \end{cases}$$

```python
def calculate_A(S_B, W, b):
    Z = np.dot(W.T, S_B) + b
    A = sigmoid(Z)
    return A.T

def sigmoid(Z):
    A = 1/(1 + np.exp(-Z))
    return A

def loss_function(Y, A):
    L = - np.sum( np.dot(Y.T, np.log(A)) + np.dot((1 - Y).T, np.log(1 - A)) ) / Y.shape[0]
    return L
```

$$\begin{cases} \dfrac{\partial L}{\partial w1} = \dfrac{\sum_{i=1}^{n}(A_i - Y_i)\cdot S_i}{n} \\ \dfrac{\partial L}{\partial w2} = \dfrac{\sum_{i=1}^{n}(A_i - Y_i)\cdot B_i}{n} \\ \dfrac{\partial L}{\partial b} = \dfrac{\sum_{i=1}^{n}(A_i - Y_i)\cdot 1}{n} \\ w1_{new} = w1_{old} - \dfrac{\partial L}{\partial w1}\cdot\alpha \\ w2_{new} = w2_{old} - \dfrac{\partial L}{\partial w2}\cdot\alpha \\ b_{new} = b_{old} - \dfrac{\partial L}{\partial b}\cdot\alpha \end{cases}$$

```python
def update_weights_bias(W, b, S_B, Y, A, learning_rate):
    dW = np.dot(S_B, (A-Y)) / S_B.shape[1] # 2x1 matrix
    db = np.sum(A - Y) / S_B.shape[1]

    W = W - dW * learning_rate
    b = b - db * learning_rate
    return W, b
```

DigiPen INSTITUTE OF TECHNOLOGY SINGAPORE

Logistic Regression Codes

```
# Logistic regression training
def train(X, Y, iters, learning_rate):

    W = np.zeros((X_train.shape[0], 1)) #initialize weights as 2x1 matrix with value 0
    b = 0

    for i in range(iters):
        A = calculate_A(X, W, b)
        L = loss_function(Y, A)
        W, b = update_weights_bias(W, b, X, Y, A, learning_rate)

        loss.append(L)
        if i == 10 or i == 1000 or i == 10000 or i >= iters-4:
            print ("iter={:d} \t W1={:f} \t W2={:f} \t b={:f} \t loss={:f}".format(i, W[0][0], W[1][0], b, L))

    return W, b
```

```
loss = []
W, b = train(X_train, Y_train, 200000, 0.01)

print("\n best W: \n", W, "\n and b:\n", b)

```

```
iter=10         W1=0.010789     W2=-0.003872    b=-0.005466     loss=0.691814
iter=1000       W1=0.941400     W2=-0.236599    b=-0.367267     loss=0.585017
iter=10000      W1=5.499376     W2=-0.487563    b=-2.271309     loss=0.288622
iter=199996     W1=19.776076    W2=-0.091501    b=-8.888023     loss=0.092960
iter=199997     W1=19.776110    W2=-0.091501    b=-8.888038     loss=0.092960
iter=199998     W1=19.776143    W2=-0.091501    b=-8.888053     loss=0.092960
iter=199999     W1=19.776177    W2=-0.091502    b=-8.888068     loss=0.092960

 best W:
 [[19.77617678]
 [-0.0915015 ]]
 and b:
 -8.888068381139373
```

```
preds_train = predict(X_train, W, b)
print(f"Accuracy on train data: {test_accuracy(preds_train, Y_train)}%")

preds_test = predict(X_test, W, b)
print(f"Accuracy on test data: {test_accuracy(preds_test, Y_test)}%")
```

```
Accuracy on train data: 98.75%
Accuracy on test data: 95.0%
```

```
sample = np.array([[1200,3]])
normalized_sample = scaler.transform(sample)
normalized_sample = normalized_sample.T

print ("For a house with size_sqft = {:d} and "
       "num_bedrooms = {:d}, the prediction is".format(sample[0][0],sample[0][1]),
       '1 (i.e., Yes, easy to sell)' if predict(normalized_sample, W, b)[0][0] == 1
       else '0 (i.e., No, not easy to sell)')
```

```
For a house with size_sqft = 1200 and num_bedrooms = 3, the prediction is 1 (i.e., Yes, easy to sell)
```

DigiPen
INSTITUTE OF TECHNOLOGY
SINGAPORE