Logistic_discrimination(X, Y, eta, max_iter): given data from two linearly seperable classes ω_1 and ω_2, trains a linear logistic discriminator to distinguish the two classes.

This function uses gradient descent with the update rule described as follows, as per presented in the slides:

$$w_i = w_i + \Delta w_i, \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \text{ where } E \text{ is the crossentropy loss}$$

$$\Delta w_j = \eta \sum_i (y_i - z_i) x_{\mathrm{ij}}, j = 1, 2, \cdots, l$$

$$\Delta w_0 = \eta \sum_i (y_i - z_i)$$

$$z_i = \sigma\left(w^T x + w_0\right)$$

Args:

- X: the input data.
- Y: the labels corresponding to X. Must be 0 and 1 encoded.
- η : desired learning rate.
- $\mathrm{max\_iter}$: maximum number of allowed iterations.

Returns:

- w: the trained weights that can perform as a logistic discriminator for X according to Y.

```
function w = logistic_discrimination(X, Y, eta, max_iter)
    if nargin<4
        if nargin < 3
            % learning rate
            eta = 0.01;
        end
        % maximum number of allowed iterations, if it takes longer than
        % this, assume the algorithm cannot converge
        max_iter = 5000;
    end
    % augment the training data with a vector of "1"s
    X_aug = [X, ones([size(X,1), 1])];
    % set initial weights to a standard normal value
    w = normrnd(0, 1, [3,1]);
    % convergence tolerance
    % if the updates are smaller than a certain amount, assume convergence
    convergence_tol = 1e-8;
    % the perceptron algorithm main loop
    iteration = 1;
    while iteration < max_iter
        delta_w = zeros([3, 1]);
        % note that because we augmented X, we don't need to separate the
        % special case for the bias term as the update rule will
        % automatically account for it.
        for j=1:3
            sum = 0;
            for i=1:size(X_aug,1)
```

```
            sum = sum + (Y(i) - logsig(w'*X_aug(i,:)'))*X_aug(i,j);
        end
        delta_w(j) = eta*sum;
    end
    if max(abs(delta_w), [], 'all') < convergence_tol
        % assume convergence
        break;
    end
    w = w + delta_w;
    iteration = iteration + 1;
  end
  fprintf('Stopped in %d steps.\n', iteration);
  disp('Final weights:');
  w
end
```