Introduction to Machine Learning

"A computer program is said to learn from experience ***E*** *with respect to some class of tasks* ***T*** *and performance measure* ***P****, if its performance at tasks in* ***T****, as measured by* ***P****, improves with experience* ***E****."* – Tom Mitchell

Experience (**E**): It is usually refers to the dataset with which the algorithm is trained. Depending on the problem statement, the nature of the dataset may vary. Some algorithms require historic data, some may require image data and some algorithms use real-time environment specific data (Reinforcement Learning). For most algorithms we perform 'training' with a subset of the collected data (training set) and 'test' the trained model with another subset of the collected data (test set). It is important that these two subsets satisfy the following conditions,

- The subsets need to be mutually exclusive and exhaustive.
- The subsets need to follow the same distribution, i.e., the nature of the data must be consistent.

Say for example, we have to predict whether a student will pass or fail the finals depending on his performances throughout the semester. We may consider a collection of performances of 10,000 students in the semester along with their final result as the dataset (Experience). The performance may include their weekly quiz scores' average, their mid-term scores, attendance, etc. We refer to these parameters as **features or attributes**. The value that we want to predict (in this case, whether the student passed or not) is referred to as the **ground truth**.

We represent the feature vector using $\boldsymbol{X}$ and the ground truth using $\boldsymbol{y}$. The aim of any machine learning algorithm is to **approximate** the function that maps $\boldsymbol{X}$ to $\boldsymbol{y}$. For the example that we mentioned above, the features and corresponding ground truths are,

$$X \;=\; \begin{bmatrix} quiz\ average \\ mid-term\ score \\ attendance \end{bmatrix} \qquad y \;=\; \begin{bmatrix} passed\ or\ not \end{bmatrix}$$

Since there are there real-valued variables in vector $\boldsymbol{X}$ and only two possible values for $\boldsymbol{y}$, we say that,

$$X \in \mathbf{R}^3 \quad \text{and} \quad y \in \{0,\ 1\}$$

$$\begin{tikzcd} {} \arrow[r, "\boldsymbol{X}"] & \boxed{f(X)} \arrow[r, "\boldsymbol{y}"] & {} \end{tikzcd}$$

Note: We can prove that there exists a hypothetical function (***f(X)***) that can successfully map $\boldsymbol{X}$ to $\boldsymbol{y}$ with no error. But it is almost impossible for a learning algorithm to approximate a function that can precisely perform the mapping. Therefore, we try to learn an **estimating** function ($\hat{\boldsymbol{f}}(\boldsymbol{X})$) that is very close the hypothetical function and provides a reasonable estimation of the ground truth.

Task (**T**): Depending on the nature of the problem statement, we decide on the kind of task that needs to be performed. There are various classes of tasks, but most of them fall under two major categories – **Classification** and **Regression**. When the expected output of the function is a discrete and finite set of values, which in most cases will be binary (for example, 'yes' or 'no') or categorical, we perform classification. Our previous example of predicting whether the student will pass or not is a classification task. In regression we aim to map the features to a real number. Therefore, as opposed to the output of classification, the output of regression will be continuous values (mostly confined to a finite range). If in our example, we intend to find the students' final scores rather than predicting whether they will pass or fail, the task would be a regression task.

Performance Measure (**P**): Algorithms that try to perform the aforementioned tasks do not learn everything in one go. Rather, they undergo an iterative learning process. After each iteration of learning, we evaluate the hypothesis of the algorithm (the function learnt in that iteration) using a performance measure. The performance measure compares the **predicted values** with the **ground truth** and gives an estimate of how close these two values are. In an ideal world, we want these two values to be equal. The difference between the predicted value and the ground truth (expected value) is referred to as **error**. The choice of performance measure depends on various factors like,

- The type of task the algorithm intends to perform.
- The specifics of the problem statement, i.e., what you want to optimize.

Our Assumptions

For the sake of simplicity and ease of visualisation, we restrict ourselves to only one feature in $\boldsymbol{X}$ and one output in $\boldsymbol{y}$. Needless to say that the ideas expressed with this primitive example can be extrapolated to any number of features and outputs. Though numerous algorithms have been proposed for both regression and classification, we will stick to two simple yet widely used algorithms – **Linear Regression** and **Logistic Regression**. We will evaluate the performance of these algorithms using separate sets performance metrics. For Linear Regression (regression task), we will use Mean Squared Error (MSE), Mean Absolute Error (MAE) and R^2 Score. For Logistic Regression (classification task), we will use binary cross-entropy as the performance measure and analyse the model's predictions based on precision, recall and F_1-score.

Linear Regression

Linear Regression falls under the category of regression. As we discussed earlier, the output of a regression task is real-valued. Therefore, we express the feature vector and the ground truth as,

$$X \in \mathbf{R}^1 \quad \text{and} \quad y \in \mathbf{R}^1$$

The linear regression model tries to establish a linear relationship between the feature $\boldsymbol{X}$ and ground truth $\boldsymbol{y}$. Therefore, the linear regression model estimates a linear function that best fits the distribution of the random variable pair $(\boldsymbol{X},\ \boldsymbol{y})$. Let us consider the following example for our discussion.

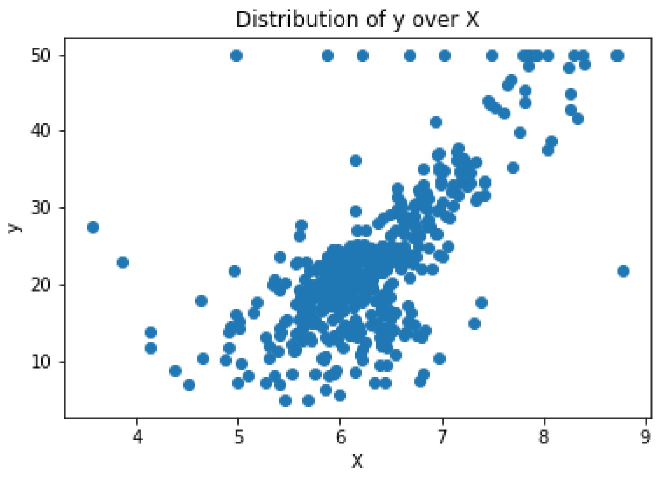


Figure 1: Distribution of y over X

Two variables $\boldsymbol{u}$ and $\boldsymbol{v}$ are said to be **linearly correlated** if a linear increase in one variable leads to linear increase in the other. Linear regression assumes that the variables $\boldsymbol{X}$ and $\boldsymbol{y}$ are linearly correlated, and tries to find the **regression line** that best fits the correlation. A general line equation in a 2D plane is given by,

$$y = wx + b$$

Thus, a line in a 2D plane is dictated by two **independent** values - $\boldsymbol{w}$ (weight) and $\boldsymbol{b}$ (bias). Therefore, the aim of the Linear Regression algorithm is to find that set of $(\boldsymbol{w},\ \boldsymbol{b})$ out of all possible such pairs defining a line that best fits all the points in the plot. It is evident that a single line cannot contain all the points in the scatter plot shown in Fig. 1. We aim, instead, to find the line of best fit. Due to this approximation we end up with a deviation from the original value, called error. Let $\hat{\boldsymbol{y}}$ be the value predicted by the Linear Regression model whose parameters are $\boldsymbol{w}$ and $\boldsymbol{b}$. According to the definition, the predicted value $\hat{\boldsymbol{y}}$ is computed as,

$$\hat{y} = wX + b \tag{1}$$

The error is calculated as the difference between the actual value (ground truth) and the predicted value,

$$\begin{aligned} error &= ground\ truth\ -\ predicted\ value \\ &= y - \hat{y} \end{aligned} \tag{2}$$

Say for example, we have $\boldsymbol{m}$ different values of $\boldsymbol{X}$ and we use them to calculate $\boldsymbol{m}$ values of $\hat{\boldsymbol{y}}$ using Eqn. (1). Then we try to calculate the total error by averaging all $\boldsymbol{m}$ errors calculated using Eqn. (2). It is possible that pairs of opposite signed errors can nullify each other while taking the average. Thus a simple mean of errors cannot give a good estimate of overall error. Therefore we define a function called **cost function** that takes the expected value ($\boldsymbol{y}$) and the predicted value ($\hat{\boldsymbol{y}}$) as inputs and estimates the deviation of the predicted value from the ground truth. Though there are many cost functions, we will stick to easiest of them all – Mean Squared Error (MSE), which is given by,

$$J(\mathbf{y},\ \hat{\mathbf{y}}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3}$$

Expanding $\hat{\boldsymbol{y}}$ using Eqn. (1),

$$J(\mathbf{y},\ w\mathbf{X} + b) = \frac{1}{m}\sum_{i=1}^{m}(y_i - (wX_i + b))^2 \tag{4}$$

In Eqn. (4) we see that the cost function $\boldsymbol{J}$ is a function of just $\boldsymbol{w}$ and $\boldsymbol{b}$, as the ground truth and the input features are already known. Thus we represent the cost function as,

$$\begin{aligned} J(w,\ b) &= \frac{1}{m}\sum_{i=1}^{m}(y_i - (wX_i + b))^2 \\ &= \frac{1}{m}\sum_{i=1}^{m} y_i^2 - 2y_i(wX_i + b) + w^2X_i^2 + 2wX_ib + b^2 \\ &= \frac{w^2}{m}\sum_{i=1}^{m} X_i^2 + \frac{2w}{m}\sum_{i=1}^{m}(y_iX_i) + \frac{2b}{m}\sum_{i=1}^{m}(y_i) + \frac{2wb}{m}\sum_{i=1}^{m}(X_i) + \frac{b^2}{m}\sum_{i=1}^{m}1 + \frac{1}{m}\sum_{i=1}^{m}y_i^2 \\ &= w^2C_1 + wC_2 + bC_3 + wbC_4 + b^2 + C \end{aligned} \tag{5}$$

From Eqn. (5) we observe that the the cost function ($\boldsymbol{J}$) is quadratic in $\boldsymbol{w}$ and $\boldsymbol{b}$. Such functions when plotted in a 3D space will result in a paraboloid.

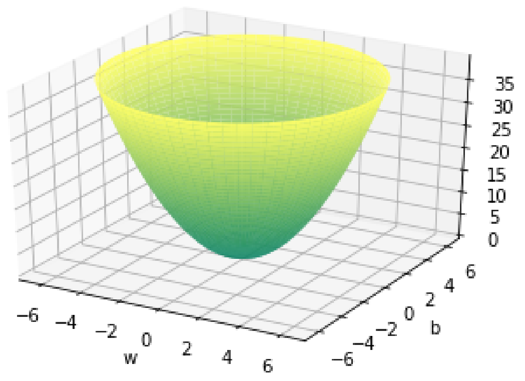


Figure 2: Cost Function J as the function of w and b

It is evident that for the algorithm to perform better, the cost function $\boldsymbol{J}$ should be minimum. The beauty of choosing MSE as the cost function is that the plot of the function is an upward paraboloid, and all upward paraboloids have one defined global minimum. A function with such well defined minima and with no saddle point (a point where the function neither increases or decreases) is called a **convex function**. The cost function chosen to evaluate the performance of the model must be a convex function.

One possible way of minimising the cost function is by differentiating Eqn. (4) partially w.r.t. to $\boldsymbol{w}$ and $\boldsymbol{b}$, and finding the optimum value of $\boldsymbol{w}$ and $\boldsymbol{b}$. We call this the **normal equation** method. But such an approach is often computationally expensive when there are more features, as the same operation is performed repeatedly to find optimum value of $\boldsymbol{w_1}$, $\boldsymbol{w_2}$,. . . $\boldsymbol{w_n}$ and $\boldsymbol{b}$. Therefore, we take an indirect approach to reach the minimum of the function − **gradient descent**.

Gradient is defined as the rate of change of a function with respect to a variable. For a function in one variable, the gradient is simply the derivative of the function w.r.t. that variable.

$$if\ y = f(x)\ then,\ gradient = \frac{d}{dx}f(x)$$

For functions in more than one variable, like our cost function, we define gradient as the partial derivative of the function w.r.t. to each variable.

$$if\ y = f(x_1, x_2, x_3)\ then,\ gradient = \nabla(y) = \begin{bmatrix} \nabla_{x_1}(y) \\ \nabla_{x_2}(y) \\ \nabla_{x_3}(y) \end{bmatrix} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \frac{\partial y}{\partial x_3} \end{bmatrix}$$

If the gradient with respect to a variable is positive then the function increases with an increase in that variable. We therefore need to reduce the value of the variable in order to reach the minimum. If the gradient is negative then the function decreases as the variable increases. Therefore the we need to increase the value of the variable to reach the minimum. In either case, we travel in the direction opposite to that of the gradient. This approach of descending towards the minimum based on the value of the gradient is called gradient descent. In general, to get the optimum value of a function, we update the variable in the opposite direction of the gradient.

$$x_i \leftarrow x_i - \frac{\partial y}{\partial(x_i)} \quad \forall i \tag{6}$$

In Linear Regression the function for which we aim to find the minimum is the cost function ($\boldsymbol{J}$). From Eqn. (4) we conclude that $\boldsymbol{J}$ is a function of two variables $\boldsymbol{w}$ and $\boldsymbol{b}$. Therefore the gradient descent algorithm dictates that the parameters $\boldsymbol{(w,\ b)}$ should be updated in the opposite direction of their gradients computed in $\boldsymbol{J}$. Therefore we arrive at the following equations.

$$w \leftarrow w - \frac{\partial J(w,b)}{\partial(w)} \qquad\qquad b \leftarrow b - \frac{\partial J(w,b)}{\partial(b)} \tag{7}$$

In practice, we note that these updates when performed for functions with sharp minima often overshoot the values of the variables from the minima. Therefore we introduce another parameter called **learning rate**, denoted by $\boldsymbol{\eta}$, to control the amount by which these values are updated. Therefore (7) is rewritten as,

$$w \leftarrow w - \eta\frac{\partial J(w,b)}{\partial(w)} \qquad\qquad b \leftarrow b - \eta\frac{\partial J(w,b)}{\partial(b)} \tag{8}$$

Eqn. (8) is the core idea of Linear Regression and many other Machine Learning and Deep Learning Algorithms. In Linear Regression we initialise $\boldsymbol{w}$ and $\boldsymbol{b}$ with random values and update them as per Eqn. (8) till we reach close to the minima. Now that we have established the fundamental relationship, we need to compute the gradients of the cost function with respect to $\boldsymbol{w}$ and $\boldsymbol{b}$ to implement the algorithm.

Computing Gradients

$$\begin{aligned}
\frac{\partial J(w,\ b)}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{m}\sum_{i=1}^{m}(y_i - (wX_i + b))^2 \\
&= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w}((wX_i + b) - y_i)^2 \\
&= \frac{2}{m}\sum_{i=1}^{m}((wX_i + b) - y_i)\frac{\partial}{\partial w}((wX_i + b) - y_i) \\
&= \frac{2}{m}\sum_{i=1}^{m}(\hat{y} - y_i)(X_i) \qquad (9)
\end{aligned}$$

Similarly we can compute the gradient of $\boldsymbol{b}$ in $\boldsymbol{J}$ which will be,

$$\frac{\partial J(w,\ b)}{\partial w} = \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i) \tag{10}$$

Now that we have established all requirements needed to implement the Linear Regression algorithm, we will list out the steps involved in the algorithm.

Algorithm 1: Linear Regression

```
Data: Input features ($\boldsymbol{X}$), ground truth ($\boldsymbol{y}$), learning rate ($\boldsymbol{\eta}$), number of steps
      ($\boldsymbol{steps}$)
Result: Optimum values of $w$ and $b$ such that $\boldsymbol{J(w,\ b)}$ is minimum
$w \leftarrow$ random value, $b \leftarrow$ random value
$i \leftarrow 1$
while $i \leq steps$ do
    $\hat{y} \leftarrow wX_j + b \quad \forall j$
    $cost \leftarrow \frac{1}{m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$
    $gradient_w = \frac{2}{m}\sum_{j=1}^{m}(\hat{y}_j - y_j)(X_j)$
    $gradient_b = \frac{2}{m}\sum_{j=1}^{m}(\hat{y}_j - y_j)$
    $w \leftarrow w - \eta \times gradient_w$
    $b \leftarrow b - \eta \times gradient_b$
    $i \leftarrow i + 1$
end
```

Performing the above algorithm for 10000 iterations with a learning rate of 0.01, the regression line that fits the distribution shown in Fig. 1 is plotted in Fig. 3.

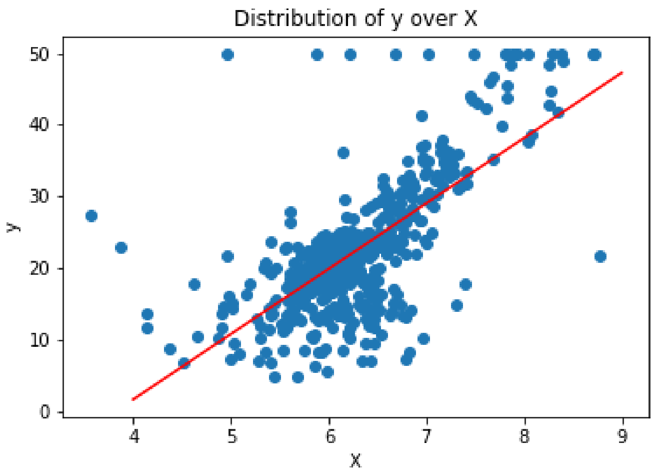
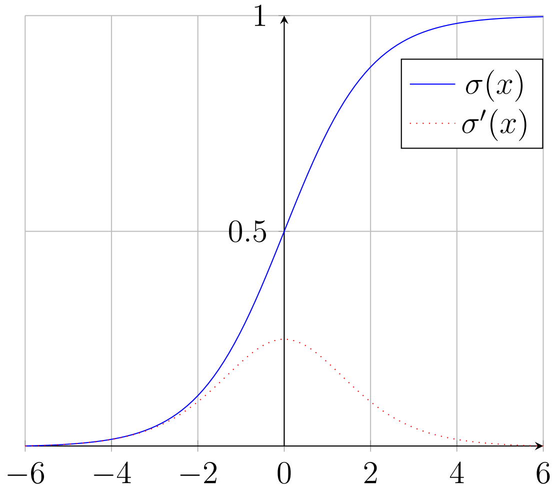


Figure 3: Regression line for the distribution

Logistic Regression

Logistic Regression is a classification algorithm used for binary classification, i.e, there are only two classes. It is similar to the Linear Regression algorithm in that these two algorithms try to learn the parameters of a linear function $(\boldsymbol{w},\ \boldsymbol{b})$ that maps $\boldsymbol{X}$ to $\boldsymbol{y}$. The only difference is the nature of the output. The classification task requires a discrete output. Therefore our estimating function $\boldsymbol{f(X)}$ should have a discrete output as well. However, the problem is that the gradient descent algorithm assumes that the cost function is differentiable at all points. Since the output of the estimating function ($\boldsymbol{\hat{y}}$) is a discrete value, we will not be able to differentiate it to get the gradients. Moreover, the linear function $\boldsymbol{wX + b}$ returns a real value which need not be in the range [0, 1]. Therefore, we need to use another function which scales down the value given by $\boldsymbol{wX + b}$ to a value in the range [0, 1]. The function should also be differentiable at all points. One such function is the **sigmoid** function, given by,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Therefore, instead of defining $\boldsymbol{\hat{y} = wX + b}$ we define the predicted value as,

$$\hat{y} = \frac{1}{1 + e^{-(wX+b)}}$$

Now let us find $\frac{\partial \hat{y}}{\partial w}$ and $\frac{\partial \hat{y}}{\partial b}$.

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{1 + e^{-(wX+b)}} \\
&= \frac{-1}{(1 + e^{-(wX+b)})^2} \times \frac{\partial}{\partial w}(1 + e^{-(wX+b)}) \\
&= \frac{-1}{(1 + e^{-(wX+b)})^2} \times e^{-(wX+b)} \times (-X) \\
&= \hat{y}(1 - \hat{y}) \times (X) \quad (\textit{substituting the value of } \hat{y}) \qquad (11)
\end{aligned}$$

Similarly,

$$\frac{\partial \hat{y}}{\partial b} = \hat{y}(1-\hat{y}) \tag{12}$$

We see that the sigmoid function is continuous throughout its domain. Therefore, we assume that if $\boldsymbol{\hat{y}_k} > 0.5$ then the example $\boldsymbol{X_k}$ is classified as 1, else it is classified as 0.

To compute the cost and update the parameters, we need an alternate cost function (rather than MSE) as the quantities that we will be computing will lie in the range [0, 1], and their differences are low. Thus the squares of their differences are much lesser. Due to these facts, MSE will not give a good estimate of the cost. We define another cost function called **Binary Cross-entropy** given by,

$$J(y,\ \hat{y}) = \frac{-1}{m}\sum_{i=1}^{M} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

Computing Gradients

$$\begin{aligned}
\frac{\partial J(y,\ \hat{y})}{\partial w} &= \frac{\partial}{\partial w}\frac{-1}{m}\sum_{i=1}^{m} y\log\hat{y} + (1-y)\log(1-\hat{y}) \\
&= \frac{-1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w}\Bigg(y\log\hat{y} + (1-y)\log(1-\hat{y})\Bigg) \\
&= \frac{-1}{m}\sum_{i=1}^{m} y\frac{\partial}{\partial w}\log\hat{y} + (1-y)\frac{\partial}{\partial w}\log(1-\hat{y}) \\
&= \frac{-1}{m}\sum_{i=1}^{m}\frac{y}{\hat{y}}\frac{\partial}{\partial w}\hat{y} - \frac{(1-y)}{(1-\hat{y})}\frac{\partial}{\partial w}\hat{y} \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(\frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}\right)\frac{\partial}{\partial w}\hat{y} \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(\frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}\right)\hat{y}(1-\hat{y})X \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(\frac{y(1-\hat{y}) - \hat{y}(1-y)}{\hat{y}(1-\hat{y})}\right)\hat{y}(1-\hat{y})X \\
&= \frac{-1}{m}\sum_{i=1}^{m}\left(\frac{y - y\hat{y} - \hat{y} + y\hat{y}}{\hat{y}(1-\hat{y})}\right)\hat{y}(1-\hat{y})X \\
&= \frac{-1}{m}\sum_{i=1}^{m}(y-\hat{y})X \\
\frac{\partial J(y,\ \hat{y})}{\partial w} &= \frac{1}{m}\sum_{i=1}^{m}(\hat{y}-y)X \qquad (13)
\end{aligned}$$

Similarly,

$$\frac{\partial J(y,\ \hat{y})}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y) \tag{14}$$

Using the gradients given by (13) and (14) we can update the parameters w and b.

```
Algorithm 2: Logistic Regression
  Data: Input feature ($\boldsymbol{X}$), ground truth ($\boldsymbol{y}$), learning rate ($\boldsymbol{\eta}$), number of steps
        ($\boldsymbol{steps}$)
  Result: Optimum values of $\boldsymbol{w}$ and $\boldsymbol{b}$ such that $\boldsymbol{J(w,\ b)}$ is minimum
  $w \leftarrow$ random value, $b \leftarrow$ random value
  $i \leftarrow 1$
  while $i \leq steps$ do
      $\hat{y} \leftarrow \sigma(wX_j + b) \quad \forall j$
      $cost \leftarrow \frac{-1}{m}\sum_{j=1}^{m} y_j \log \hat{y_j} + (1 - y_j)\log(1 - \hat{y_j})$
      $gradient_w = \frac{2}{m}\sum_{j=1}^{m}(\hat{y_j} - y_j)(X_j)$
      $gradient_b = \frac{2}{m}\sum_{j=1}^{m}(\hat{y_j} - y_j)$
      $w \leftarrow w - \eta \times gradient_w$
      $b \leftarrow b - \eta \times gradient_b$
      $i \leftarrow i + 1$
  end
```