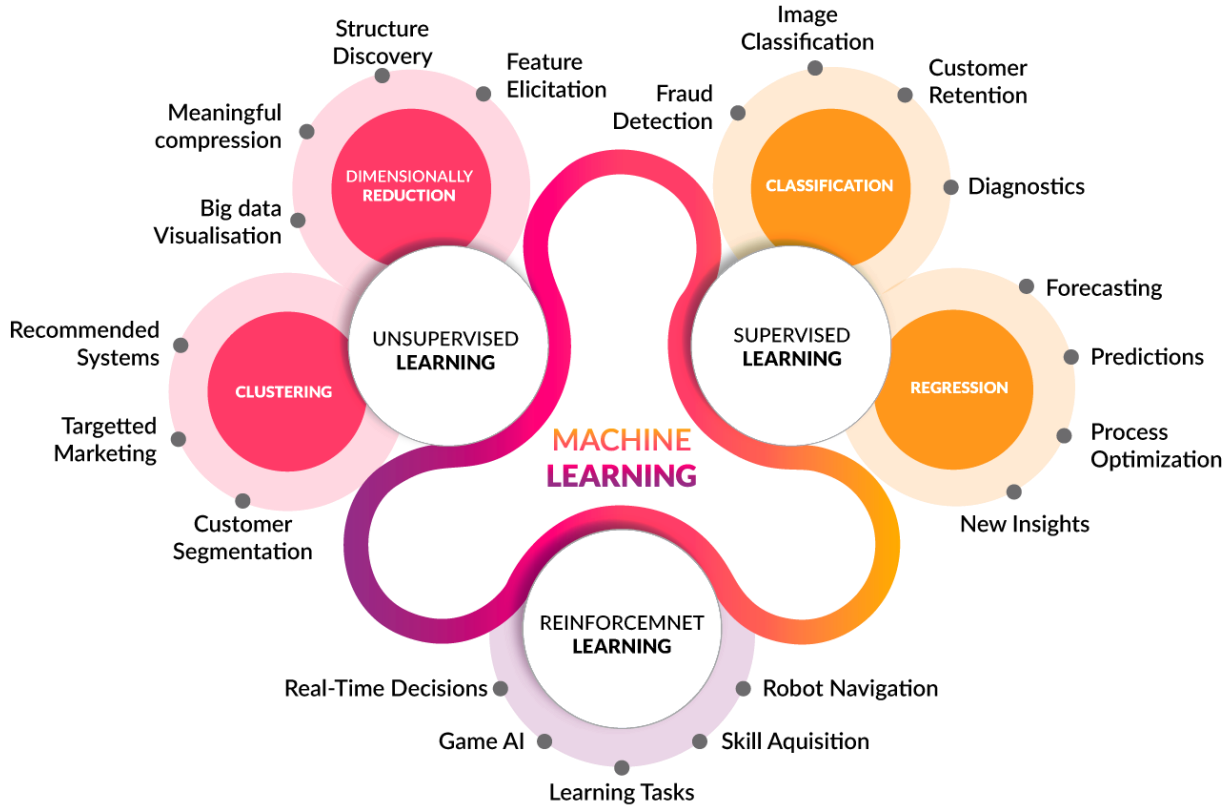


Supervised Learning

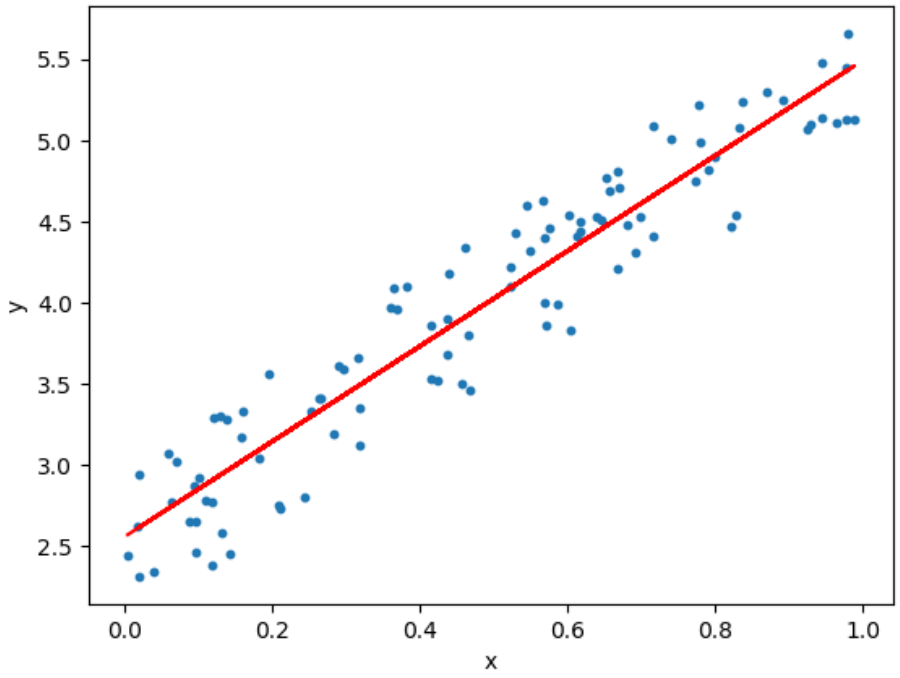
UNIT 2

Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data.

One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.

For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.



Linear Regression (contd.)

Linear regression is a simple yet powerful and mostly used algorithm in data science.

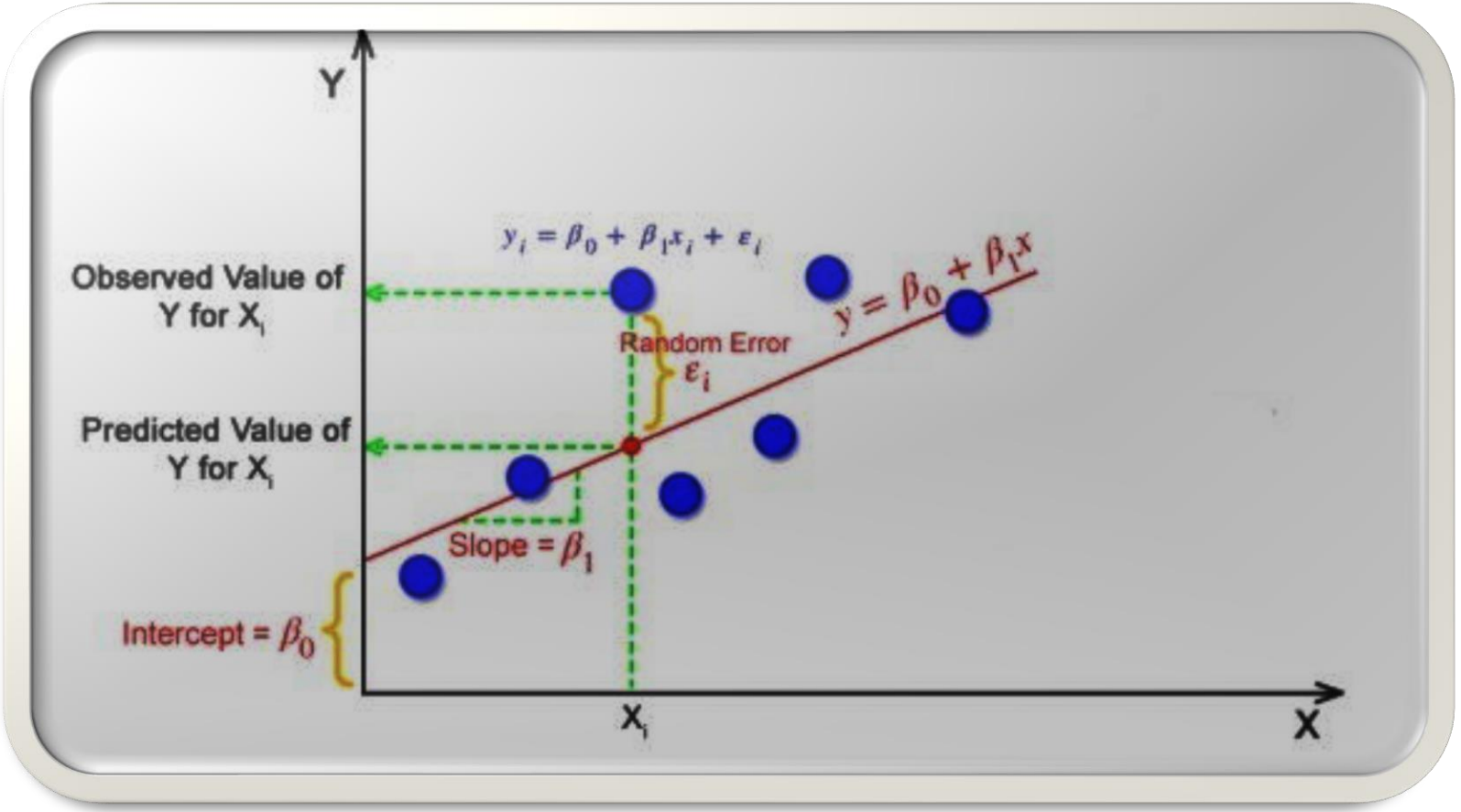
Linear regression shows the linear relationship between the independent(predictor) variable i.e. X-axis and the dependent(output) variable i.e. Y-axis, called linear regression.

If there is a single input variable X(independent variable), such linear regression is called simple linear regression.

The above graph presents the linear relationship between the output(y) variable and predictor(X) variables.

The red line is referred to as the best fit straight line.

Linear Regression (contd.)



Linear Regression (contd.)

The goal of the linear regression algorithm is to get the best values for β_0 and β_1 to find the best fit line.

In simple terms, the best fit line is a line that fits the given scatter plot in the best way.

The best fit line is a line that has the least error which means the error between predicted values and actual values should be minimum.

Linear Regression (contd.)

Random Error(Residuals)

In regression, the difference between the observed value of the dependent variable(yi) and the predicted value is called the **residuals**.

$$\boldsymbol{e} = \boldsymbol{y}predicted - \boldsymbol{y}_{actual}$$

Then we can define the RSS (Residual Sum of Squares) as

$$RSS = e_1^2 + e_2^2 + \dots\dots\dots\dots\dots\dots\dots\dots\dots + e_m^2$$

In Linear Regression, generally **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the $y_{predicted}$ and yi.

$$MSE = 1/m \sum_{i=1}^{m} \left(y_i - (\beta_0 + B_1 x)\right)^2$$

We use different optimization techniques to minimize the error or RSS. Some of them are ordinary least square method, gradient descent etc.

Gradient Descent

We use linear regression to predict the dependent continuous variable y on the basis of independent X. It assumes the relationship between independent and dependent variables to be linear as such:

$$h(x) = \hat{y}_i = w_0 + w_1 x_1 + \ldots + w_n x_n = \sum_{j=1}^{n} w_j x_j$$

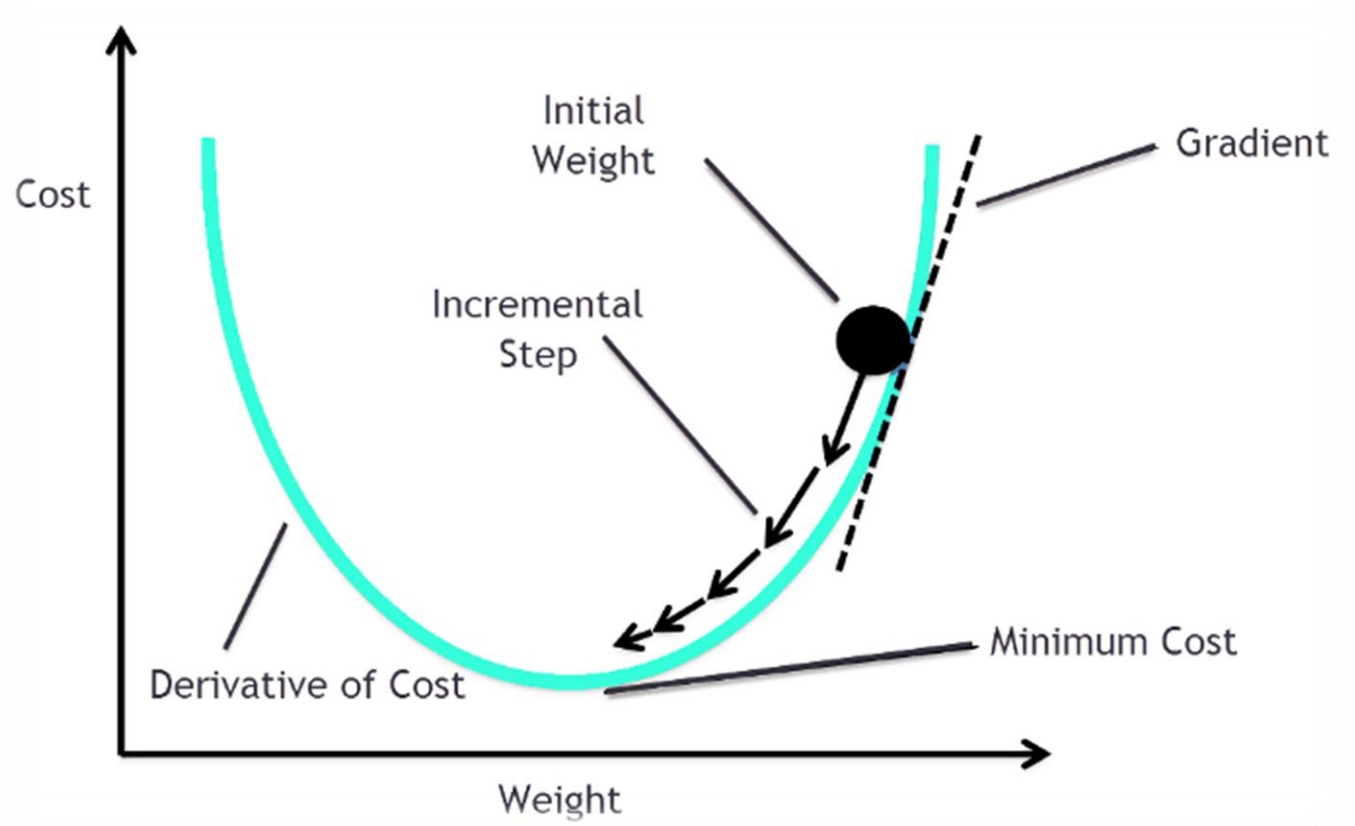
Gradient Descent is the optimization algorithms that optimize the cost function(objective function) to reach the optimal minimal solution.

To find the optimum solution we need to reduce the cost function(MSE) for all data points.

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)} \right)^2$$

This is done by updating the values of $w_0 \ldots w_1$ iteratively until we get an optimal solution.

Gradient Descent



Gradient Descent (contd.)

We used the update rule as:

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w)$$

where α is the learning rate.

And, considering only one training example (x, y), we compute the derivate as:

$$\frac{\partial}{\partial w_j} J(w)$$
$$= \frac{\partial}{\partial w_j} \frac{1}{2} (h(x) - y)^2$$
$$= \frac{1}{2} \frac{\partial}{\partial w_j} (h(x) - y)^2$$
$$= \frac{1}{2} \frac{\partial (h(x) - y)^2}{\partial (h(x) - y)} \frac{\partial (h(x) - y)}{\partial w_j}$$

$$\frac{\partial}{\partial w_j} J(w)$$
$$= \frac{1}{2} \frac{\partial (h(x) - y)^2}{\partial (h(x) - y)} \frac{\partial (h(x) - y)}{\partial w_j}$$
$$= \frac{1}{2} 2\,(\mathrm{h(x)} - \mathrm{y}). \frac{\partial}{\partial w_j} \sum_{i=0}^{n} (w_j x_j - y)$$
$$= (h(x) - y).x_j$$

Gradient Descent (contd.)

Now, we can rewrite the update rule as:

$$w_j = w_j - \alpha(h(x) - y).x_j$$

Which gives,

$$w_j = w_j + \alpha(y - h(x)).x_j$$

And this is the required form for the weight update. This rule is called the **LMS (*Least Mean Square*) update rule** and is also known as the **Windrow-Hoff learning rule**.

Gradient Descent (contd.)

This update has several properties:

1. The update has direction in opposite of gradient i.e. we always move towards the opposite of the gradient.
2. The magnitude of the update is proportional to the error term $\left(y - h(x)\right)$ i.e. the lesser the error is smaller the update on parameters and larger change to parameters occur if our prediction has larger error.

Types of Gradient Descent

1. **Batch Gradient Descent**

   In this type of gradient descent, we compute the gradient term for all the training examples and then we update the parameters with respect to the gradient of all training observations.

   It computes the error of each training example within training observations but are summed up once before update is performed.

   This gradient descent has to scan through the entire training set before taking a single step.

   When training set is very large, it is costly operation to take a single step for convergence.

   This is computationally efficient algorithm and it produces stable gradient and stable convergence.

   Downside, it also requires entire training observations be in memory and available throughout the model training.

Types of Gradient Descent (contd.)

1. Batch Gradient Descent (contd.)

Its implementation is given as:

repeat until convergence {

$$w_j = w_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h\left(x^{(i)}\right) \right) . x_j^{(i)}$$

}

Types of Gradient Descent (contd.)

2. **Stochastic Gradient Descent**

   In this type of gradient descent, we repeatedly run through the training set and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example.

   Often, Stochastic Gradient Descent gets β close to minimum much faster compared to batch gradient descent.

   It is preferred when the training set is large.

   Due to the frequent update, the Gradient descend will oscillates more resulting noisy gradients and each update is computationally expensive compared to batch gradient descent.

Types of Gradient Descent (contd.)

2. Stochastic Gradient Descent (contd.)

Its implementation is given as:

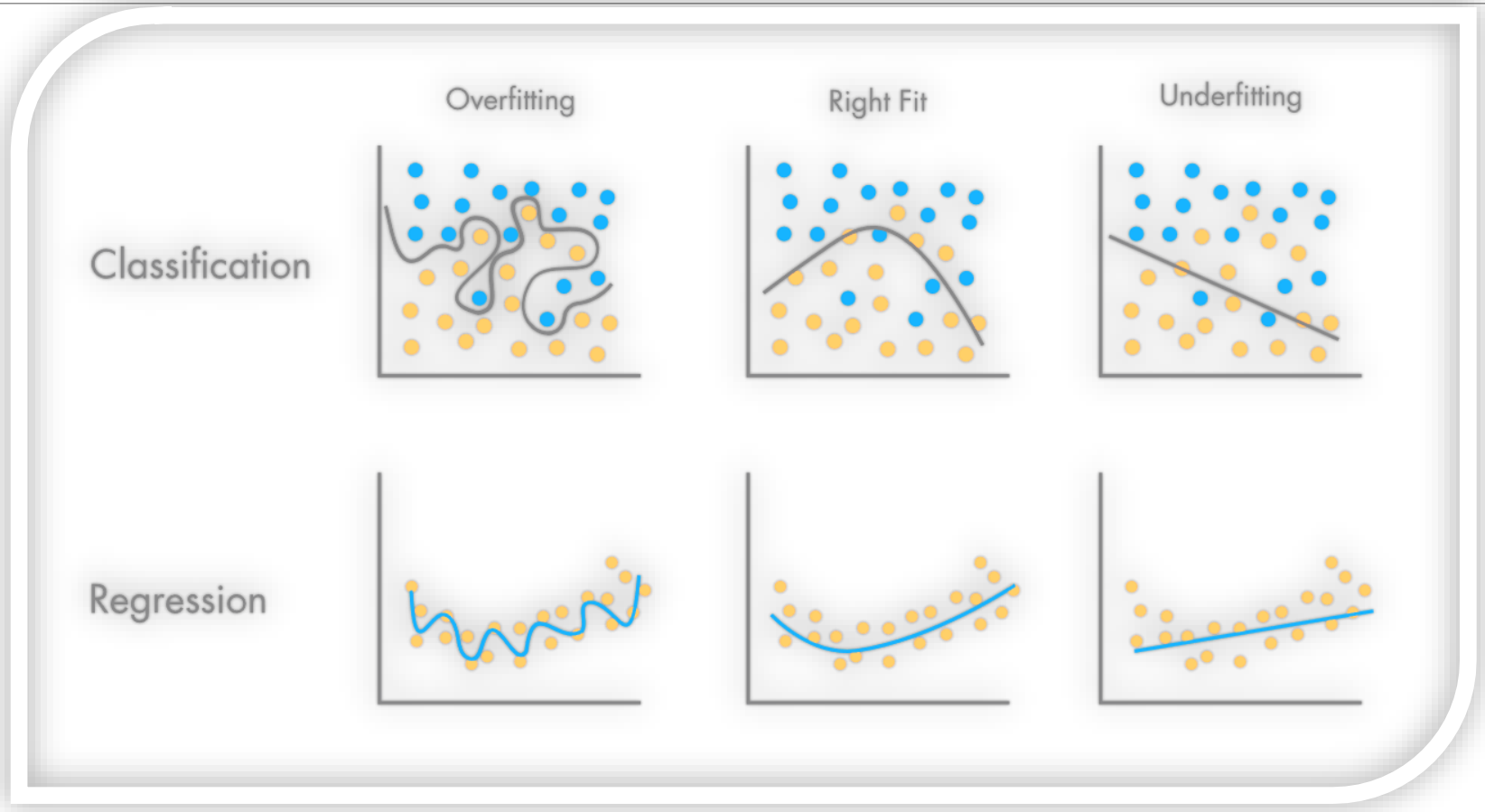
```
Loop for every j-th feature {

    For every training example i from 1 to m {
```

$$\beta_j = \beta_j + \alpha\left(y^{(i)} - h_\beta\left(x^{(i)}\right)\right).x_j^{(i)}$$

```
    }
}
```

Overfitting and Underfitting



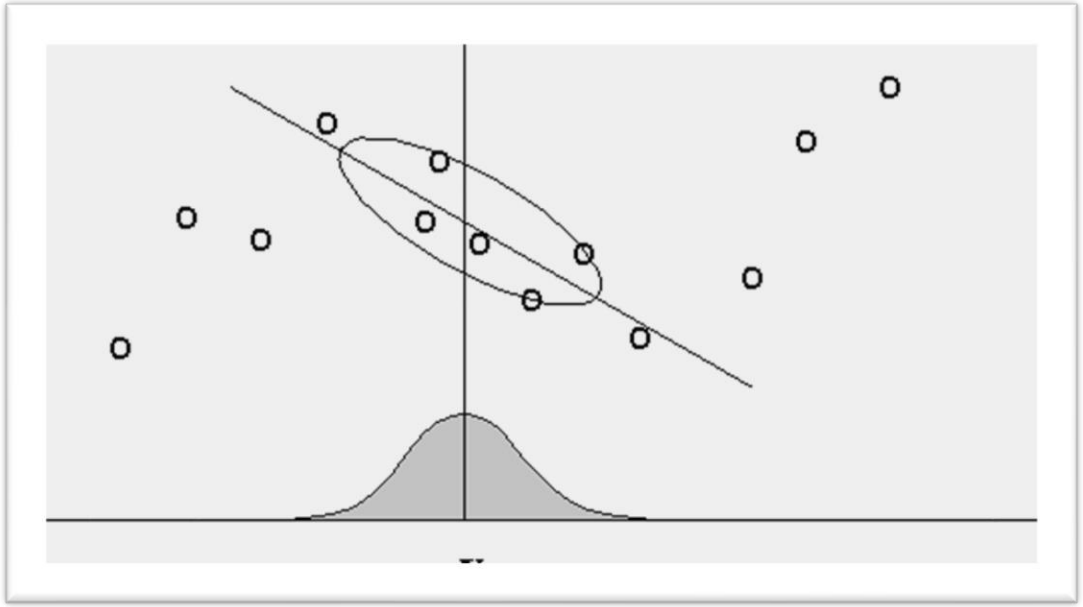
Locally Weighted Regression

Locally weighted regression is a non-parametric algorithm i.e. non-parametric algorithm make any assumptions about underlying data.

Also, non-parametric algorithm doesn't learn a fixed set of parameters as in linear regression algorithm.

It is also known as memory based method that performs regression around a point of interest and uses a training data that are local to that point only.

In locally weighted regression, points are weighted by proximity to the current x in question using a kernel. A regression is then computed using the weighted points.

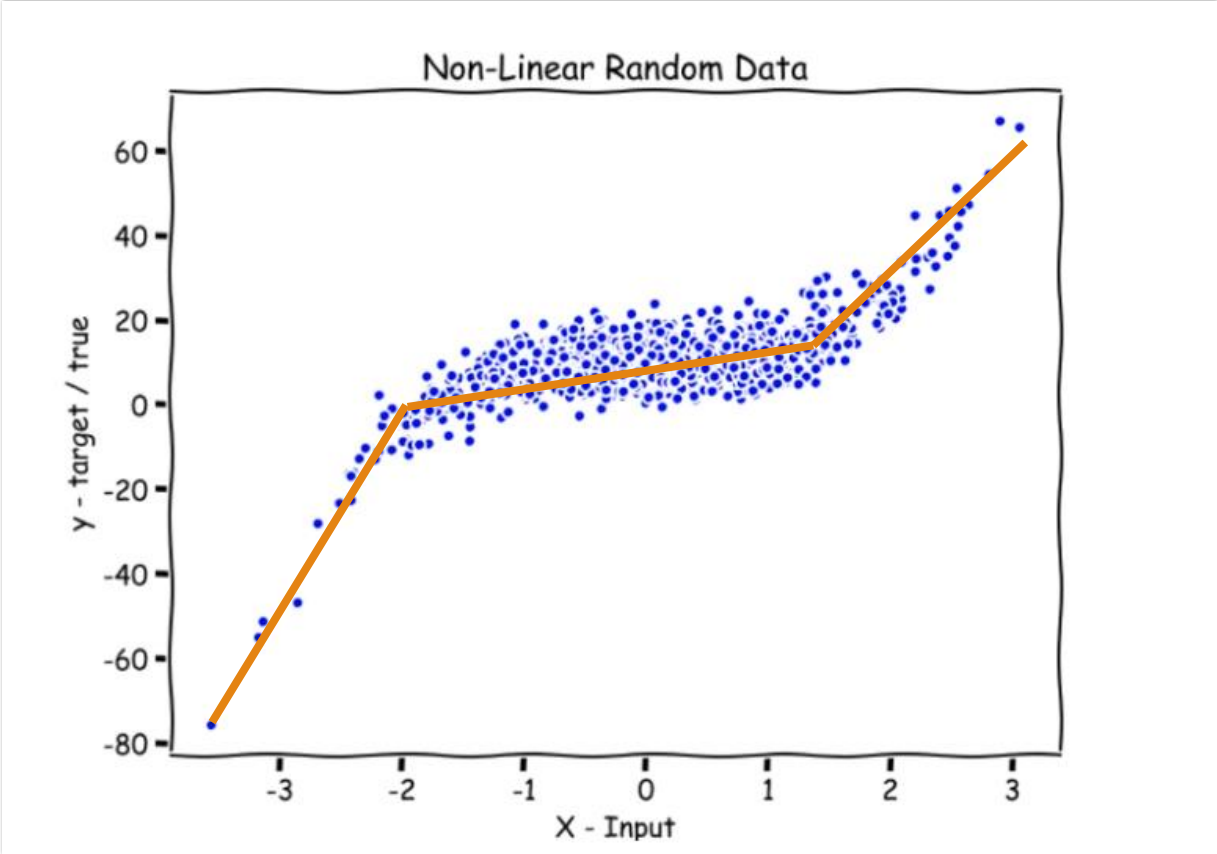


Locally Weighted Regression (contd.)

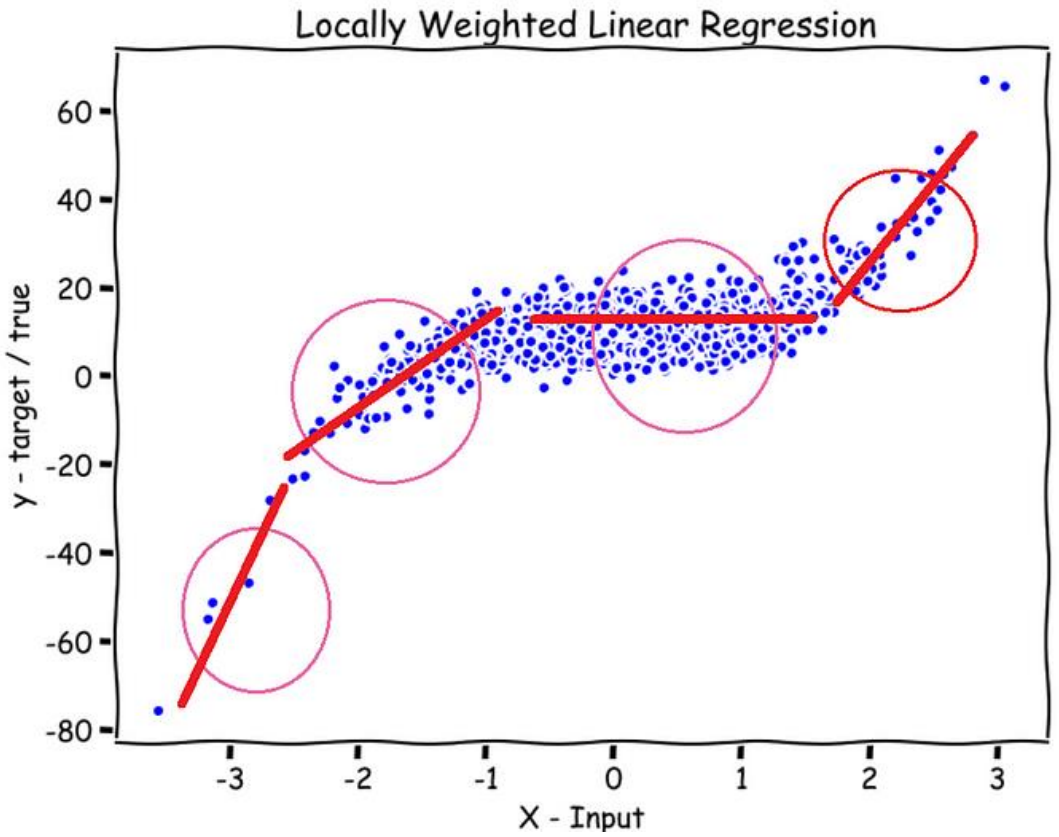
How does it works?

- LWLR is a non-parametric regression technique that fits a linear regression model to a dataset by giving more weight to nearby data points.
- The weights assigned to each training data point are inversely proportional to their distance from the query point.
- Training data points that are closer to the query point will have a higher weight and contribute more to the linear regression model.
- LWLR is useful when a global linear model does not well-capture the relationship between the input and output variables. The goal is to capture local patterns in the data.

Locally Weighted Regression (contd.)



Locally Weighted Regression (contd.)



In locally weighted regression, points are weighted by proximity to the current **x** in question using a kernel. A regression is then computed using the weighted points.

Locally Weighted Regression (contd.)

In linear regression we train model that learns w_j from the given training data such that during the training we tend to minimize $\sum_i \left(y^{(i)} - h(x^{(i)})\right)^2$.

Whereas, with the locally weighted linear regression algorithm, we tend to minimize $\sum_i \omega^{(i)} \left(y^{(i)} - h(x^{(i)})\right)^2$ during the training. That is cost function is given as:

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} \omega^{(i)} \left(h(x^{(i)}) - y^{(i)}\right)^2$$

Here, $\omega^{(i)}{}'s$ are non-negative valued called weights (*Don't get confused with w_j which is actually a coefficient and defines the weight of features*) which refers to the value of particular data points in dataset.

Locally Weighted Regression (contd.)

Intuitively, if $\omega^{(i)}$ is large for a particular value of i, then in picking ω, we will train our model to make $\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$ small. Similarly, if $\omega^{(i)}$ is small, then the $\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$ error term will be pretty much ignored in the fit.

A common choice for the weights is

$$\omega^{(i)} = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

where τ is the bandwidth parameter which depends on the particular point x at which we're trying to evaluate x.

Locally Weighted Regression (contd.)

The bandwidth parameter τ decides the width of the neighborhood you should look into to fit the local straight line.

In other words, τ controls how quickly the weight of a training example $x^{(i)}$ falls off with distance from the query point x.

Depending on the choice of τ, we make a choice among fatter or thinner bell-shaped curve, which cause us to look in a bigger or narrower window respectively, which in turn decides the number of nearby examples to use in order to fit the straight line.

τ has an effect on overfitting and underfitting:

- If τ is too broad, you end up over-smoothing the data leading to underfitting.
- If τ is too thin, you end up fitting a very jagged fit to the data leading to overfitting. T is a hyper-parameter of the locally weighted regression algorithm.

Locally Weighted Regression (contd.)

There's a huge literature on choosing τ . For example, instead of the bell-shaped curve, a triangle shaped function can be used so your weights actually go to zero outside some small range. So there are many versions of this algorithm.

Moreover, if $|x^{(i)} - x|$ is small, then $\omega^{(i)}$ is close to 1; and if $|x^{(i)} - x|$ is large, then $\omega^{(i)}$ is small.

Hence, ω is chosen giving much higher weight to the data points close to the query point x than the data points farther to the query point.

*Note: Although the formula of ω takes the form that is cosmetically similar to pdf of Gaussian distribution, they do not directly have anything to do with Gaussians, and in particular they are not random variables normally distributed as well.

Logistic Regression

Logistic Regression is a binary classification problem where we use linear line to separate between two known categories of interest called classes.

This is similar to the regression problem, except that the values of y we wan tot predict takes discrete values.

As logistic regression is binary classification, we have two classes where all the data points belongs, i.e. $y \epsilon \{0, 1\}$.

For instance, we are trying to classify email to spam and ham classes through spam classifier. $x^{(i)}$ refers to the i-th training dataset that either belongs to ham class or spam class . Thus, $y^{(i)}$ has value 1 if it is spam, 0 otherwise.

Note that for given $x^{(i)}$, corresponding $y^{(i)}$ is given in data and called as **label**.

Logistic Regression (contd.)

Using linear regression, we predict h(x) as:

$$h(x) = \hat{y} = \sum w_j x_j = w^T x$$

where the value of $\hat{y}$ is continuous.

Let's say instead of predicting continuous values for $\hat{y}$, we are taking probabilities P. Thus the value of P must also lie within 0 and 1. But h(x) consists of value ranging from – Inf to + Inf.

So instead of just taking probabilities we take odds:

i.e. $\text{odds} = \frac{p}{1-p}$

Logistic Regression (contd.)

The **odds** are defined as the probability that the event will occur divided by the probability that the event will not occur. Thus, Odds are nothing but the ratio of the probability of success and probability of failure.

The value of odds lies between 0 to +ve infinity.

And, If the odds are high (million to one), the probability is almost 1.00. If the odds are tiny (one to a million), the probability is tiny, almost zero.

To convert from a probability to odds, divide the probability by one minus that probability. So if the probability is 10% or 0.10 , then the odds are 0.1/0.9 or ‘1 to 9’ or 0.111.

To convert from odds to a probability, divide the odds by one plus the odds. So to convert odds of 1/9 to a probability, divide 1/9 by 10/9 to obtain the probability of 0.10.

Logistic Regression (contd.)

But, the problem here with the ***odds*** is that the range is restricted as values of odds lies between to Infinity and we don't want a restricted range because if we do so then our correlation will decrease.

By restricting the range it is difficult to model a variable. Hence, in order to control this we take the log of odds which has a range from (-∞,+∞).

So, we get,

$$\log\left(\frac{p}{1-p}\right) = \sum_{j=1}^{n} w_j x_j = w^T x$$

Now, taking exponentiation on both side

$$\exp\left(\log\left(\frac{p}{1-p}\right)\right) = \exp(w^T x)$$

Logistic Regression (contd.)

$$\frac{p}{1-p} = \exp(w^T x)$$

$$\Rightarrow\ p = \exp(w^T x) - p.\exp(w^T x)$$

$$\Rightarrow 1 = \frac{\exp(w^T x)}{p} - \exp(w^T x)$$

$$\Rightarrow 1 + \exp(w^T x) = \frac{\exp(w^T x)}{p}$$

$$\Rightarrow p(1 + \exp(w^T x)) = \exp(w^T x)$$

$$\Rightarrow\ p = \frac{\exp(w^T x)}{(1+\exp(w^T x))}$$

Now, dividing both side by $\exp(w^T x)$

$$\Rightarrow p = \frac{\exp(w^T x)/\exp(w^T x)}{(1+\exp(w^T x))/\exp(w^T x)}$$

$$\Rightarrow p = \frac{1}{1/\exp(w^T x)+1}$$

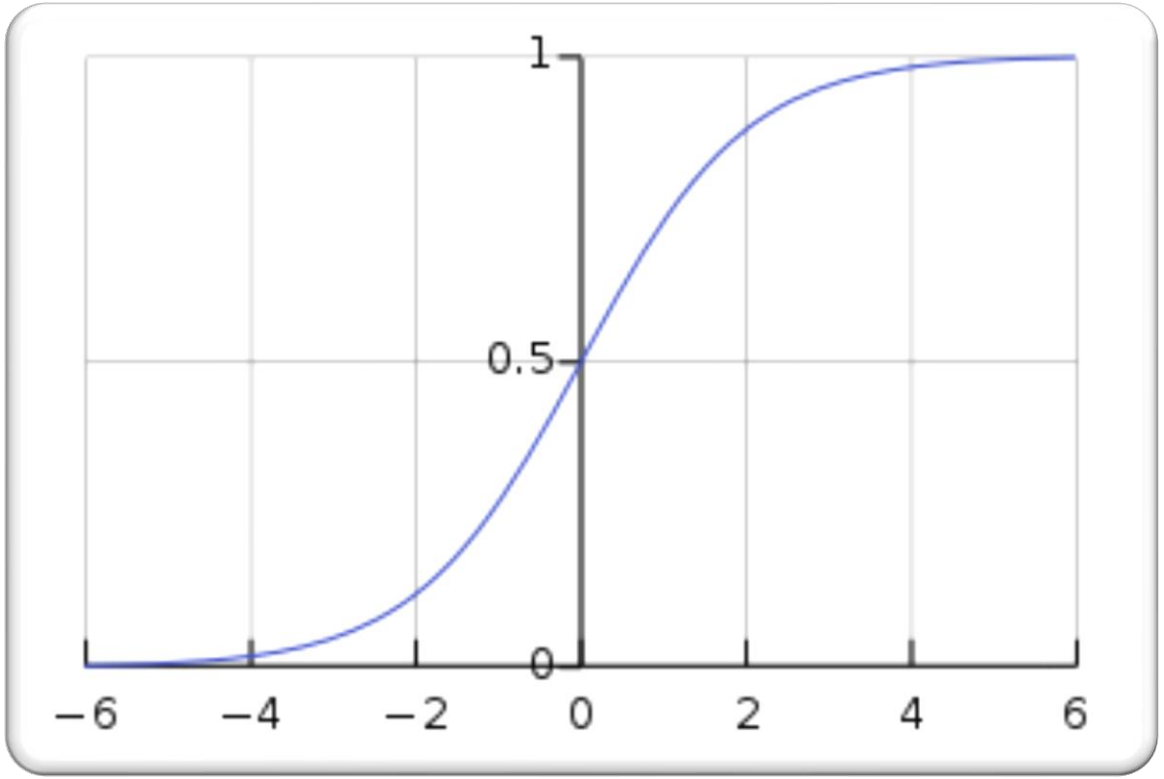
$$\Rightarrow p = \frac{1}{\exp(-w^T x)+1}$$

$$\Rightarrow p = \frac{1}{1+\exp(-z)} \text{ where } z = w^T x$$

This gives us the **sigmoid function** or the logistic function which is the function of z or $w^T x$.

Logistic Regression (contd.)

Graphically, the sigmoid function is



Logistic Regression (contd.)

From above, we have our hypothesis for logistic function as:

$$h(x) = g(z) = \frac{1}{1 + \exp(-z)}$$

Here, as $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. This now shows the value of $h(x)$ is bounded within 0 and 1.

We now get the equation for logistic function which we can use for logistic regression. But how do we fit the model? How do we determine the weight coefficient of our model?

Logistic Regression (contd.)

Given x, let the probability that the given data point x belongs to class 1

$$P(y = 1|\, x; w) = h(x)$$
$$P(y = 0|\, x; w) = 1 - h(x)$$

Or we can write as:

$$p(y|x; w) = (h(x))^{y}\big(1 - h(x)\big)^{1-y}$$

Lets assume that the m training examples were generated independently, we can the write the likelihood of the parameters as:

$$L(w) = p(y|X; w) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; w) = \prod_{i=1}^{m} (h\big(x^{(i)}\big))^{y^{(i)}}\big(1 - h(x)\big)^{1-y^{(i)}}$$

Logistic Regression (contd.)

Instead of maximizing $L(w)$, we can also maximize any strictly increasing function of $L(w)$. In particular, to make our derivation simple we maximize the likelihood:

$$l(w) = \log L(w) = \log\left(\prod_{i=1}^{m} (h(x^{(i)}))^{y^{(i)}} (1 - h(x))^{1-y^{(i)}}\right)$$

$$\Rightarrow l(w) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})$$

This is also the form of loss function commonly known as Cross Entropy Loss function, commonly used across classification problem.

But how do we maximize the likelihood?

Answer to this is, similar to gradient descent used in linear regression, we can use gradient ascent since we need to maximize the likelihood. And this is given as:

$$w_j = w_j + \alpha \frac{\partial \mathrm{L}}{\partial w_j}$$

Logistic Regression (contd.)

Then, let's determine the derivate of log likelihood and as before we consider only one training example to ease our derivation:

$$\frac{\partial\, l(w)}{\partial w_j} = \frac{\partial}{\partial w_j}[y \log h(x) + (1-y)\log(1-h(x)]$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \frac{\partial}{\partial w_j}\left[y \log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right) + (1-y)\log(1-g(\boldsymbol{w}^T\boldsymbol{x})\right]$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = y\frac{\partial}{\partial w_j}\log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right) + (1-y)\frac{\partial}{\partial w_j}\log\left(1 - g(\boldsymbol{w}^T\boldsymbol{x})\right)$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = y\frac{\partial}{\partial g(\boldsymbol{w}^T\boldsymbol{x})}\log\left(g(\boldsymbol{w}^T\boldsymbol{x})\right)\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j} + (1-y)\frac{\partial \log\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}{\partial\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}\frac{\partial\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}{\partial g(\boldsymbol{w}^T\boldsymbol{x})}\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j}$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[y\frac{1}{g(\boldsymbol{w}^T\boldsymbol{x})} + (1-y)\frac{1}{1-g(\boldsymbol{w}^T\boldsymbol{x})}(0-1)\right]\frac{\partial\, g(\boldsymbol{w}^T\boldsymbol{x})}{\partial w_j}$$

** $\boldsymbol{w}^T\boldsymbol{x}$ is a vector dot product form of $\boldsymbol{w}$ and $\boldsymbol{x}$ vector and equals to $\sum w_j x_j$*

Logistic Regression (contd.)

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[\frac{y}{g(\boldsymbol{w}^T\boldsymbol{x})} - \frac{(1-y)}{1-g(\boldsymbol{w}^T\boldsymbol{x})}\right]\frac{\partial}{\partial w_j} g(\boldsymbol{w}^T\boldsymbol{x})$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[\frac{y\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)-(1-y)g(\boldsymbol{w}^T\boldsymbol{x})}{g(\boldsymbol{w}^T\boldsymbol{x})\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right)}\right] g(\boldsymbol{w}^T\boldsymbol{x})\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right) x_j \quad (*How?)$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[y\left(1-g(\boldsymbol{w}^T\boldsymbol{x})\right) - (1-y)g(\boldsymbol{w}^T\boldsymbol{x})\right]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = \left[y\left(1-h(x)\right) - (1-y)h(x)\right]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = [y - yh(x) - h(x) + yh(x)]x_j$$

$$\Rightarrow \frac{\partial\, l}{\partial w_j} = [y - h(x)]x_j$$

Which is the required form.

$$g'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \frac{\partial\left(\frac{1}{1+e^{-z}}\right)}{\partial(1+e^{-z})}\frac{\partial(1+e^{-z})}{\partial\, e^{-z}}\frac{\partial\, e^{-z}}{\partial z}$$

$$= -\frac{1}{(1+e^{-z})^2}(0+1)(-e^{-z})$$

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{e^{-z}}{1+e^{-z}}\frac{1}{1+e^{-z}}$$

$$= \frac{-1+1+e^{-z}}{1+e^{-z}}\frac{1}{1+e^{-z}}$$

$$= \left(\frac{-1}{1+e^{-z}} + \frac{1+e^{-z}}{1+e^{-z}}\right)\frac{1}{1+e^{-z}}$$

$$= (-g(z)+1)\, g(z) = g(z)(1-g(z))$$

Logistic Regression (contd.)

Then, we can rewrite our update rule given above as:

$$w_j = w_j + \alpha(y^{(i)} - h\left(x^{(i)}\right)x_j^{(i)}$$

Which is the stochastic form of the gradient ascent update rule.

Perceptron Learning Algorithm

Consider modifying the logistic regression to "force" it to produce either 0 or 1 exactly as an output. Then, we can express it as a threshold function as:

$$g(z) = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; if \; z < 0 \end{cases}$$

And using this definition along with the update rule as:

$$w_j = w_j + \alpha \left(y^{(i)} - h\left(x^{(i)}\right) \right) x_j^{(i)}$$

We get the perceptron learning algorithm which used to be considered as the model for human's brain neuron in early days.

Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about – basically, logistic regression, it is actually a very different type of algorithm; in particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.