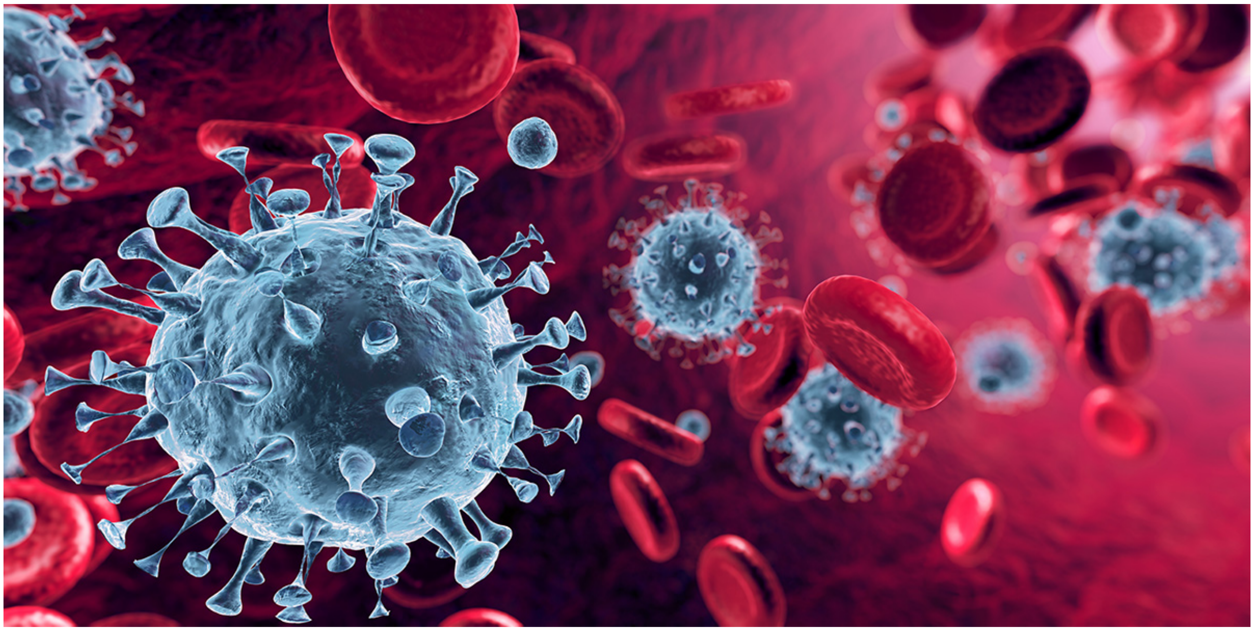
Logistic Regression Algorithm

Using Breast Cancer Data



The Intuition

Logistic regression is similiar to simple and multiple regression. The difference is it is a classifier of binary data instead of predicting continuous variables. Furthermore, the logistic regression model uses probability to predict the dependent variable.

Important Formulas :

$$\hat{y}_i = \sigma(w^T x_i + b)$$

$$L_{ce}(\hat{y}_i, y_i) = -[y_i \; log \; \hat{y}_i + (1 - y_i) \; log(1 - \hat{y}_i)]$$

$$Cost(w, b) = \frac{1}{N} \sum L_{ce}(\hat{y}_i, y_i)$$

$$gradient = \frac{\partial l_{ce}(w, b)}{\partial w_i} = [\sigma(w^T x_i + b) - y] x_i$$

$$Bias = [\sigma(w^T x_i + b) - y]$$

Code

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [3]:

```
# Read the data file
data = pd.read_csv('../../../_resources/data/cancer.csv')

# Seperate by dependent and independent variables
x = data.iloc[:, [2, 3]].values
y = data.iloc[:, [1]].values

# Change y variables to 1 or 0
for i in range (0,len(y)):
    if(y[i] == 'M'):
        y[i] = 1
    else:
        y[i] = 0


data.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

5 rows × 33 columns

Split the data

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0
```

Create Logistic Regression Class

In [ ]:

```
class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=100000, fit_intercept=True, ve
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.lr * gradient

            if(self.verbose == True and i % 10000 == 0):
                z = np.dot(X, self.theta)
                h = self.__sigmoid(z)
                print(f'loss: {self.__loss(h, y)} \t')

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X, threshold):
        return self.predict_prob(X) >= threshold
```

In [ ]:

```
# Needed Variables
n_iter = 1000
theta = np.zeros(x.shape[1])
lr = 0.01

model = LogisticRegression(lr=0.1, num_iter=300000)
%time model.fit(x_train, y_train)


preds = model.predict(x_test, .5)
# accuracy
(preds == y_test).mean()

```

```
In [ ]:
```

```
In [ ]:
```