In [2]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [3]:

```
data = load_breast_cancer()
X,y = load_breast_cancer(return_X_y=True)
```

In [4]:

```
col = data['feature_names']
X = pd.DataFrame(X, columns=col)
X.head()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

5 rows × 30 columns

In [5]:

```
y = pd.DataFrame(y, columns=['target'])
y
```

Out[5]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
564	0
565	0
566	0
567	0
568	1

569 rows × 1 columns

In [6]:

```
X.shape
```

Out[6]:

```
(569, 30)
```

In [7]:

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns=col)

bias = pd.DataFrame(np.ones((X.shape[0],1)), columns=['bias'])
X = pd.concat((bias,X), axis=1)
```

In [8]:

```
df = pd.concat((X,y),axis=1)
df.head()
```

Out[8]:

	bias	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mea symmetr
0	1.0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515
1	1.0	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392
2	1.0	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685
3	1.0	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383
4	1.0	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.00956

5 rows × 32 columns

In [9]:

```
df['target_new'] = df['target'].replace(0,-1)
df['target_new'].unique()
```

Out[9]:

```
array([-1,  1])
```

In [10]:

```
y_new = df['target_new']
```

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y_new,
test_size=0.2, random_state=0)
```

In [12]:

```
W = np.zeros((X.shape[1]))
W
```

Out[12]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [13]:

```
X_train = X_train.to_numpy()
y_train = y_train.to_numpy()
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()
```

Perceptron Algorithm

In [14]:

```
for t in range(500):
    for i in range(X_train.shape[0]):
        affine = np.dot(np.array(X_train[i]),W)
        if affine >= 0:
            y_pred = 1
        else:
            y_pred = -1
        if y_pred != y_train[i]:
            W += y_train[i]*X_train[i]
```

In [15]:

```
def predict(X):
    linear_output = np.dot(X, W)
    return np.where(linear_output >= 0, 1, -1)


predictions = predict(X_test)
accuracy = np.mean(predictions == y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
Accuracy: 96.49%
```

Gradient Descent Algorithm

In [16]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
X_train = X_train.to_numpy()
y_train = y_train.to_numpy()
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()
```

In [48]:

```
learning_rate= 0.1
max_iter = 1000
weights = np.zeros((X_train.shape[1],1))
weights
```

Out[48]:

```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
```

```
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

In [49]:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

In [51]:

```
for t in range(max_iter):
    summation = 0
    linear = np.dot(X_train, weights)
    y_pred = sigmoid(linear)

    summation += np.dot(X_train.T, (y_train-y_pred))

    dw = -summation/X_train.shape[0]

    weights -= learning_rate*dw
```

In [57]:

```
def predict(X):
    linear_output = np.dot(X, weights)
    linear_output = np.array(list(map(sigmoid, linear_output)))
    return np.where(linear_output >= 0.5, 1, 0.)

predictions = predict(X_test)
accuracy = np.mean(predictions == y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 96.49%
```