**Logistic Regression**

Logistic Regression is a popular algorithm used for supervised machine learning to predict binary output.Its uses the sigmoid curves for prediction rather than a single line as in linear regression. This help in distinguish the non linear relationship between the features and target.

I am biyam khaiju. I have made a logistic regression model without using the in-built libraries like sklearn / tensorflow.

Here is the step to step code break-down

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

Here numpy is the library used for working with array.

Matplotlib is used to create different types of plot for data visualization. Though it’s a simple program so we don’t use plotting here.

Seaborn is also used for data visualization.

def sigmoid(z):

    return 1 / (1 + np.exp(-z))

Here we define a function called sigmoid and pass the parameter z and return a statement 1/(1+np.exp(-z)) whose value will be stored in the parameter z. Here this statement is the used to predict the sigmoid graph. Also the np.exp is expressed as the exponential.

class LogisticRegression:

    def \_\_init\_\_(self, lr=0.1, epochs=1000):

        self.lr = lr

        self.epochs = epochs

        self.weights = None

        self.bias = None

In this code we have defined a class named LogisticRegression

And defined a function for the iteration process and assuming weights and bias to be zero.

    def fit(self, X, y):

        n\_samples, n\_features = X.shape

        self.weights = np.zeros(n\_features)

        self.bias = 0

Another function is defined as named as fit and self,X and y are the parameters. Here X.shape is given as the 2 dimension array as per my data mentioned at the last and n\_sample stores the number of rows and n\_features stores the number of columns.

And intially the weight and the bias are zero.

        for epoch in range(self.epochs):

            linear\_model = np.dot(X, self.weights) + self.bias

            y\_predicted = sigmoid(linear\_model)

Above we have given epoch value as 1000 so it will iterate 1000 times to obtain the best value for weight and bias. Due to the initial value of weight and bias are zero the linear\_model will be zero and y\_predicted will be 1/2 for first iteration.

            dw = (1 / n\_samples) \* np.dot(X.T, (y\_predicted - y))

            db = (1 / n\_samples) \* np.sum(y\_predicted - y)

In simple terms, dw and db are the cost function which helps to determine the weight and bias required as per the values and helps to increase the accuracy. It loops 1000 times for the best w and b values.

            self.weights -= self.lr \* dw

            self.bias -= self.lr \* db

This code basically updates the weight and bias after each iteration. Here lr is the learning which we can modify as per we need.

    def predict(self, X):

        linear\_model = np.dot(X, self.weights) + self.bias

        y\_predicted = sigmoid(linear\_model)

        return [1 if i > 0.5 else 0 for i in y\_predicted]

Now, after the training of weight and bias the actual prediction part starts. We have new function called predict and parameters self and X. The code is simply as above but if the y\_predicted is greater than 0.5 or we can say greater than 50%

It will return 1 else it will return zero. This is because the values we need can only outcomes as 0 and 1. Upto here we are 90% done and finished our training part.

X = np.array([[22, 25000], [25, 32000], [47, 70000], [52, 80000], [46, 65000], [56, 90000], [28, 40000]])

y = np.array([0, 0, 1, 1, 1, 1, 0])

model = LogisticRegression(lr=0.1, epochs=1000)

model.fit(X, y)

predictions = model.predict(X)

print("Predictions:", predictions)

Now, we have our own data which we can give it to our model to predict,the model call the LogisticRegression again which we trained above and fits the value of X and y there. Now model.predict(X) predicts the possible outputs for given X features and displays it.

This is the LogisticRegression Logic behind the scikitlearn in-built library.