# **LOGISTIC REGRESSION**

### **Aim**

To implement a logistic regression model.

### **Problem description**

This project involves implementing logistic regression from scratch on a dataset related to customer churn.Logistic regression is a supervised Learning technique predicting the categorical dependent variable using a given set of independent variables

### **Algorithm**

#### **1) Import the required libraries**

Import the python libraries numpy,pandas

**2)Define the functions required to implement logistic regression**

**3) Load the dataset**

Load the dataset **“**Telco-Customer-Churn.csv**”**

#### **4) Preprocessing the dataset**

#### **5) Create the Logistic regression model**

#### **5) Train the Logistic regression model using the training data**

#### **6) Testing the Logistic regression model using the test data**

### **Program code/ Pseudocode**

import pandas as pd

import numpy as np

import time

def sigmoid(X, weight):

z = np.dot(X, weight)

return 1 / (1 + np.exp(-z))

def loss(h, y):

return (-y \* np.log(h) - (1 - y) \* np.log(1 - h)).mean()

def gradient\_descent(X, h, y):

return np.dot(X.T, (h - y)) / y.shape[0]

def update\_weight\_loss(weight, learning\_rate, gradient):

return weight - learning\_rate \* gradient

def log\_likelihood(x, y, weights):

z = np.dot(x, weights)

ll = np.sum( y\*z - np.log(1 + np.exp(z)) )

return ll

data = pd.read\_csv(r'C:\ardhra\Logistic\_regression\WA\_Fn-UseC\_-Telco-Customer-Churn.csv')

print("Dataset size")

print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))

print("Columns and data types")

pd.DataFrame(data.dtypes).rename(columns = {0:'dtype'})

df = data.copy()

df['class'] = df['Churn'].apply(lambda x : 1 if x == "Yes" else 0)

# features will be saved as X and our target will be saved as y

X = df[['tenure','MonthlyCharges']].copy()

X2 = df[['tenure','MonthlyCharges']].copy()

y = df['class'].copy()

start\_time = time.time()

num\_iter = 100000

intercept = np.ones((X.shape[0], 1))

X = np.concatenate((intercept, X), axis=1)

theta = np.zeros(X.shape[1])

for i in range(num\_iter):

h = sigmoid(X, theta)

gradient = gradient\_descent(X, h, y)

theta = update\_weight\_loss(theta, 0.1, gradient)

print("Training time (Log Reg using Gradient descent):" + str(time.time() - start\_time) + " seconds")

print("Learning rate: {}\nIteration: {}".format(0.1, num\_iter))

result = sigmoid(X, theta)

f = pd.DataFrame(np.around(result, decimals=6)).join(y)

f['pred'] = f[0].apply(lambda x : 0 if x < 0.5 else 1)

print("Accuracy (Loss minimization):")

f.loc[f['pred']==f['class']].shape[0] / f.shape[0] \* 100

start\_time = time.time()

num\_iter = 100000

intercept2 = np.ones((X2.shape[0], 1))

X2 = np.concatenate((intercept2, X2), axis=1)

theta2 = np.zeros(X2.shape[1])

for i in range(num\_iter):

h2 = sigmoid(X2, theta2)

gradient2 = gradient\_ascent(X2, h2, y) #np.dot(X.T, (h - y)) / y.size

theta2 = update\_weight\_mle(theta2, 0.1, gradient2)

print("Training time (Log Reg using MLE):" + str(time.time() - start\_time) + "seconds")

print("Learning rate: {}\nIteration: {}".format(0.1, num\_iter))

result2 = sigmoid(X2, theta2)

print("Accuracy (Maximum Likelihood Estimation):")

f2 = pd.DataFrame(result2).join(y)

f2.loc[f2[0]==f2['class']].shape[0] / f2.shape[0] \* 100

### **Result**

Logistic regression modelis successfully implemented with 73.4% accuracy.