The AI-ML and Data Science Club of IITH

Logistic Regression

Contents

Compiled by : Srinith

1 What is Logistic Regression?

Logistic Regression is one of the **supervised learning**[1] models of Machine Learning which takes data labeled into different classes as input for training , and then for test images it finds the probabilites for it to be in those different classes (discrete values).In other words , we can say that it predicts catergorical labels for a given test image.

2 How Logistic regression works?

Let us consider binary-classification at the moment . Now we can give classes based on a value say y ($y = 0$ and $y = 1$). So , The model first estimates the real-valued output (a confidence factor) using regression and then uses a decision rule to classify the test image based on the probability associated with it(a confidence factor) into either of the classes (class 0 or class 1).

In the case of multi-class classification , we implement multiple logistic regression classifiers , one for each of the K classes in the training dataset.In each of those classifiers we treat one class as (say y=0) samples and treat all the remaining classes into y=1 samples respectively and find the probabilities.Then at last , we pick the class to which the image gets the highest probability.

We define the **Hypothesis** similar to Linear Regression but with an additional Sigmoid Function wrapped around it, so that the value which we get from hypothesis is always in between 0 and 1 (any other function other than sigmoid can be taken which brings the value to between 0 and 1), then a **Cost/Loss function** that gives the error in the Hypothesis while an algorithm is used to reduce the Cost function.

2.1 Hypothesis

Let N be the number of features and M be the number of data set points in the data.
And let the dataset X is defined as follows

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} \in \mathbb{R}^{M \times N}$$

$$x_i = \begin{bmatrix} x_i^{(1)} & x_i^{(2)} & \cdots & x_i^{(N)} \end{bmatrix} \in \mathbb{R}^{1 \times N}$$

$$y_i \in \mathbb{R}$$

where x_i denote the i^{th} data point and $x_i^{(j)}$ denotes the j^{th} feature of the i^{th} data point.
Let the Weights/Parameters/Regression Coefficients be represented by θ as follows

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix}$$

[1]Machines are trained with the data that is **tagged** with the correct output, which is used to predict the output

Let the values we get from the hypothesis function for all the data set points be represented by vector Y and bias values be represenetd by b.

$$Y = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_M} \end{bmatrix} \in \mathbb{R}^{M \times 1}$$

Hypothesis of Logistic Regression is defined as,

$$\hat{y_i} = \sigma(\langle x_i, \theta \rangle + b)$$

where $\sigma(x)$ is the sigmoid function $\left(\dfrac{1}{1+e^{-x}}\right)$ and $\langle \cdot, \cdot \rangle$ represents dot product between two vectors.

3 Cost/Loss Function

Let us take there to be two classes with labels y=0 and y=1. Then we have ,

$$p(y=1|X;\theta) = \frac{1}{1+e^{-z}} = h_\theta(x)$$
$$p(y=0|X;\theta) = 1 - \frac{1}{1+e^{-z}} = 1 - h_\theta(x)$$

where z = $\langle X, \theta \rangle$ + B and
y can take only two values 0,1. So ,

$$p(y|X;\theta) = (h_\theta(x))^y (1-h_\theta(x))^{1-y}$$

Now, we would want to find the weights θ that maximizes the log-likelihood l. Likelihood $L(\theta)$ is ,

$$\begin{aligned} L(\theta) &= p(y|X;\theta) \\ &= \prod_{i=1}^{M} p(y^{(i)}|X^{(i)};\theta) \\ &= \prod_{i=1}^{M} (h_\theta(x^{(i)}))^{y^{(i)}} (1-h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

where $x^{(i)}, y^{(i)}$ denote the observed values of X and Y in the ith training sample

$$\begin{aligned} l(\theta) &= \log(L(\theta)) \\ &= \sum_{i=1}^{M} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) \end{aligned}$$

We would want to maximize the log likelihood , which will be taken as the cost function , multiplying by a factor of 1/M doesnt change anything , so

$$J(\theta) = \frac{1}{M} \sum_{i=1}^{M} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$

Where M is the number of samples

4 Algorithms to maximize the Cost Function

One of the algorithms used in Logistic Regression to maximize the Cost function is Gradient Ascent Algorithm.

4.1 Gradient Aescent

In Gradient Ascent, we move in the direction of the slope , which is attained by finding the gradient of the Cost Function concerning each parameter and then updating the parameter iteratively.

$$\theta^{(j)} = \theta^{(j)} + \alpha \frac{\partial}{\partial \theta^{(j)}} J(\theta)$$

where α is the Learning rate. Now, the learning rate decides how big the step should be taken in that direction.

Differentiating the loss function for a single sample ,

$$\frac{\partial}{\partial \theta^{(j)}} J(\theta) = (y - h_\theta(x))x_j$$

Hence,

$$\theta^{(j)} = \theta^{(j)} + \alpha(y - h_\theta(x))x_j$$

For multiple samples,

$$\implies \theta^{(j)} = \theta^{(j)} - \frac{\alpha}{M} \sum_{i=1}^{M} (h_\theta(x^{(i)}) - y^{(i)}).x_j^{(i)}$$

5 Regularization

One of the major drawbacks of logistic regression is that it cant be used for linearly seperable data , it overfits for very high dimension and sparse data . In order to avoid this , we add a regularizer term $\lambda \mod w^2$ to our cost function. This will help control of the weight vector because the norm of the weight vector does not become very large.

6 Questions

Subjective :

1. Is the decision boundary linear or non linear in logistic regression?
2. Is Logistic Regression affected by Outliers?
3. Name three methods by which we can increase the accuracy of logistic regression.
4. Why isn't least squared error considered in Logistic Regression ?
5. What is the significance of the bias term b ?

Objective :

1) What happens when we train a non-regularized logistic regression model on linearly separable data ?

   a) The Parameters in the vector w would approach ∞
   b) The Parameters in the vector w would approach ∞
   c) The Parameters in the vector w would approach 0
   d) None of the above

2) Suppose that we train two logistic regression models on same data by initializing the weights of one model to all zeros and other to all ones. Which of the models will perform better on test data ?

   a) The model with initial weights as all zeros
   b) The model with initial weights as all ones
   c) Both models will perform similarly
   d) None of the above

3) Suppose we have a set of training inputs $(x_1, y_1), ..., (x_n, y_n)$ and we train a logistic regression classifier on this data. Now if we flip the labels on all the training inputs i.e; we change y_i to $1 - y_i$ for all i. What would happen to the predicted output/probability of assigning label 1 for each training sample ?

   a) Each probability p_i will become $1 - p_i$
   b) Each probability p_i will be halved
   c) The parameters in the vector w would approach 0
   d) None of the above

7 Logistic Regression with python

A Logistic Regression model is built from scratch by only using `numpy` , `pandas` and `matplotlib` libraries.

```
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import math
```

```
def loading_data():
    data_set = pd.read_csv('dataset.csv')

    training_data = data_set.sample(frac=0.75,random_state=25)
    testing_data = data_set.drop(training_data.index)

    train_y = training_data['quality']
    test_y = training_data['quality']
    del training_data['quality']
    del testing_data['quality']
    training_data_normal = (training_data - training_data.mean())/
       training_data.std()
    testing_data_normal = (testing_data - testing_data.mean())/
       testing_data.std()

    training_data = training_data_normal.to_numpy()
    testing_data = testing_data_normal.to_numpy()
    train_y = train_y.to_numpy()
    test_y = test_y.to_numpy()

    return (training_data,train_y),(testing_data,test_y)
```

```
(train_X,train_Y),(test_X,test_Y) = loading_data()
```

```
class LogisticRegressor(object):
  def __init__(self) :
    self.M = train_X.shape[0]
    self.N = train_X.shape[1]
    self.weights = [np.random.randn(self.N,1)]
    self.bias = np.random.randn()
    self.test_size = test_X.shape[0]

  def SGD(self,epochs,lr):
    for j in range(epochs):
      self.weights = self.weights + (lr/self.M)*(self.loss_deri())
      print("Epoch : "+ str(j) + " Loss value : " + str(self.
         loss_function()) + " Accuracy : " + str(self.accuracy()))

  def hypo_function(self,train_entry):
    z = self.bias + np.dot(train_entry,self.weights)[0][0]
    return 1/(1+np.exp(-z))

  def loss_function(self):
    loss = 0
    for i in range(self.M):
      loss = loss + train_Y[i]*(math.log(self.hypo_function(
         train_X[i]))) + (1-train_Y[i])*(math.log(self.
         hypo_function(train_X[i])))

    loss = loss/self.M
    return loss

  def loss_deri(self):
    loss_deri = np.zeros([self.N,1])
    for i in range(self.M):
      loss_deri = loss_deri + (train_Y[i] - self.hypo_function(
         train_X[i]))*(np.transpose(train_X[i]))
    return loss_deri

  def accuracy(self):
    correct = 0
    curr = 0
    for i in range(self.test_size):
          if(self.hypo_function(test_X[i]) >= 0.5):
            curr = 1
          else:
            curr = 0
          if(curr == test_Y[i]):
            correct = correct+1

        return correct/self.test_size
```

```
Model = LogisticRegressor()
```

```
Model.SGD(epochs=100,lr=0.01)
```