Logitic Regression and Gradient Descent
Logic and Python tutorial

Sophie Marchand

May 2020

1 Shortcut

The logistic regression is used to solve binary classification problem by modelling the probability p of belonging to the class 0 (or 1) through logarithm transformation and linear combination of input variables such as (x_1, x_2) as follow:

$$ln(\frac{p}{1-p}) = a + b_1 * x_1 + b_2 * x_2 = -f \Leftrightarrow p = \frac{e^{-f}}{1+e^{-f}}$$

Also, we can estimate the coefficients (a, b_1, b_2) of the logistic regression through the gradient descent with, for example, the cost function C leading to the coefficient update for $c \in (a, b_1, b_2)$ with η the learning rate and i the index of the set of the input and output values $(x_1, x_2, y)_i$:

$$C(i) = \frac{1}{2}(p - y_i)^2 \text{ leading to } c := c + \eta * \frac{\partial y}{\partial c} * (p - y_i) * p * (1 - p)$$

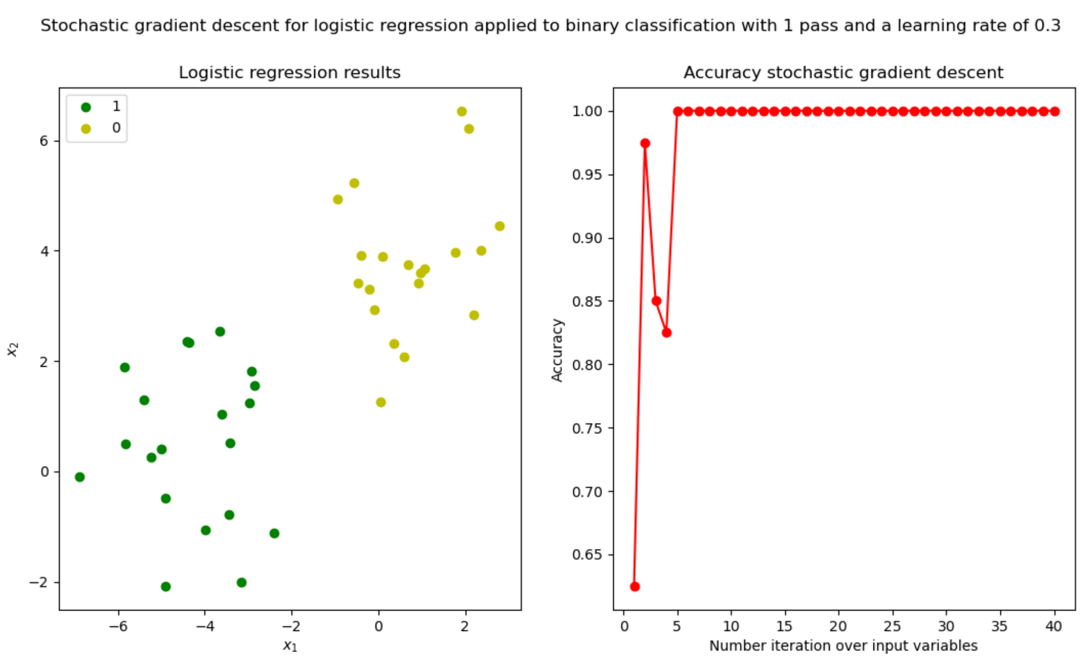

Figure 1: Output of the tutorial in section 3

2 Logic details

We present here the intuition behind the logistic regression algorithm and its resolution through stochastic gradient descent. The logistic regression is used to model the odds of belonging to the class 1 within a binary classification problem having the classes 0 and 1. We note p the probability to belong to the class 1, also the odds to belong to this class are defined as:

$$odds_1 = \frac{p}{1-p} \tag{1}$$

The idea of the logistic regression is to represent $odds_1$ as a linear function of the input variables through a logarithmic transformation as presented in equation (2). For this demonstration, we have two input variables (x_1, x_2) and the linear coefficients are noted (a, b_1, b_2).

$$ln(odds_1) = ln(\frac{p}{1-p}) = a + b_1 * x_1 + b_2 * x_2 \tag{2}$$

By rearranging the equation (2), we obtain the equation (3) where $f = -(a + b_1 * x_1 + b_2 * x_2)$ is the predicted output variable.

$$p = \frac{e^{a+b_1*x_1+b_2*x_2}}{1 + e^{a+b_1*x_1+b_2*x_2}} = \frac{e^{-f}}{1+e^{-f}} \tag{3}$$

We remark that the form of expression (3) resembles the logistic function form $\frac{1}{1+e^{-value}}$ and that is why, this method is called logistic regression. To compute the coefficients (a, b_1, b_2), we use the stochastic gradient descent with the following cost function (4) and derivative (5). We note c the considered coefficient among (a, b_1, b_2) and y_i the output variable associated to the input variables $(x_{1,i}, x_{2,i})$.

$$C(i) = \frac{1}{2}(p - y_i)^2 \tag{4}$$

$$\begin{aligned}\frac{\partial C(i)}{\partial c} = (p-y_i) * \frac{\partial p}{\partial c} = (p-y_i) * \frac{\partial(\frac{e^{-f}}{1+e^{-f}})}{\partial c} = (p-y_i) * \frac{e^{-f}}{(1+e^{-f})^2} * -\frac{\partial y}{\partial c} \\ = -\frac{\partial y}{\partial c} * (p-y_i) * \frac{e^{-f}}{1+e^{-f}} * (1 - \frac{e^{-f}}{1+e^{-f}}) = -\frac{\partial y}{\partial c} * (p-y_i) * p * (1-p)\end{aligned} \tag{5}$$

According to the gradient descent equations (see GitHub) and the equation (5), we have the following expressions for the coefficients update of the logistic regression with η the learning rate:

$$\begin{cases} a := a - \eta\frac{\partial C(i)}{\partial a} = a - \eta * (p-y_i) * p * (1-p) \\ b_1 := b_1 - \eta\frac{\partial C(i)}{\partial b_1} = b_1 - \eta * (p-y_i) * p * (1-p) * x_{1,i} \\ b_2 := b_2 - \eta\frac{\partial C(i)}{\partial b_2} = b_2 - \eta * (p-y_i) * p * (1-p) * x_{2,i} \end{cases} \tag{6}$$

Finally, using the stochastic gradient descent with 1-to-10 passes based on the equations (6), we will obtain the optimal coefficients (a, b_1, b_2) of the logistic regression. In order to identify the class of the input variables (x_1, x_2), we can define a threshold p_{th} such as:

$$\begin{cases} \text{class } = 1 \text{ for } p \geq p_{th} \\ \text{class } = 0 \text{ for } p < p_{th} \end{cases} \tag{7}$$

3 Python tutorial

The code source displayed below can be found on GitHub under Python_LogisticRegressionGradientDescent

```python
"""
Tutorial Logistic Regression with Stochastic Gradient Descent

Author: Sophie Marchand
"""
import matplotlib.pyplot as plt
import numpy as np
import sklearn.datasets as skd


# Function initialization parameters, update parameters and compute error
def init_parameters():
    return 0, 0, 0


def randomized_training_data(x_1, x_2, y):
    index_array = np.arange(len(x_1))
    np.random.shuffle(index_array)
    return x_1[index_array], x_2[index_array], y[index_array]


def update_parameters_stochastic(a, b_1, b_2, x_1, x_2, y, learning_rate, iteration):
    x_1_it = x_1[iteration]
    x_2_it = x_2[iteration]
    y_it = y[iteration]
    f = -(a + b_1 * x_1_it + b_2 * x_2_it)
    p = np.exp(-f) / (1 + np.exp(-f))
    a_update = a - learning_rate * (p - y_it) * p * (1 - p)
    b_1_update = b_1 - learning_rate * (p - y_it) * p * (1 - p) * x_1_it
    b_2_update = b_2 - learning_rate * (p - y_it) * p * (1 - p) * x_2_it
    return a_update, b_1_update, b_2_update


def compute_accuracy(a, b_1, b_2, x_1, x_2, y, p_th):
    f = -(a + b_1 * x_1 + b_2 * x_2)
    p_pred = np.exp(-f) / (1 + np.exp(-f))
    y_pred = (p_pred > p_th).astype(int)
    accuracy_pred = sum(y_pred == y) / len(y)
    return accuracy_pred


# Create data for binary classification with two input variables
X, y = skd.make_blobs(n_samples=40, n_features=2, centers=2,
                      cluster_std=1.2, random_state=3)
x_1 = np.asarray([X[i, 0] for i in range(0, len(X))])
x_2 = np.asarray([X[i, 1] for i in range(0, len(X))])
p_th = 0.5
learning_rate = 0.3
number_batch = 1
accuracy = []

# Workflow stochastic gradient descent
a, b_1, b_2 = init_parameters()
for batch in range(number_batch):
    x_1_rand, x_2_rand, y_rand = randomized_training_data(x_1, x_2, y)
    for iteration in range(len(x_1_rand)):
        a_update, b_1_update, b_2_update = \
            update_parameters_stochastic(a, b_1, b_2, x_1_rand, x_2_rand, y_rand,
    learning_rate, iteration)
        a, b_1, b_2 = a_update, b_1_update, b_2_update
        accuracy.append(compute_accuracy(a, b_1, b_2, x_1, x_2, y, p_th))
a_final, b_1_final, b_2_final = a, b_1, b_2

# Print results
```

```python
f = -(a_final + b_1_final * x_1 + b_2_final * x_2)
p_pred = np.exp(-f) / (1 + np.exp(-f))
y_pred = (p_pred > p_th).astype(int)

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Stochastic gradient descent for logistic regression applied to binary
    classification with '
             + str(number_batch) + ' pass and a learning rate of ' + str(learning_rate),
    fontsize=12)
label_legend = [0, 1]
for color in ['g', 'y']:
    if color == 'g':
        index = np.where(y_pred == 1)
        label = label_legend[1]
    else:
        index = np.where(y_pred == 0)
        label = label_legend[0]
    ax1.scatter(x_1[index], x_2[index], c=color, label=label)
ax1.set(xlabel='$x_{1}$', ylabel='$x_{2}$', title='Logistic regression results')
ax1.legend()
ax2.plot(range(1, len(accuracy) + 1), accuracy, 'ro-')
ax2.set(xlabel='Number iteration over input variables', ylabel='Accuracy',
        title='Accuracy stochastic gradient descent')
plt.show()
```