华南理工大學

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author: Wu kun

Supervisor:
Mingkui Tan or Qingyao Wu

Student ID：201721045480

Grade:
Undergraduate or Graduate

December 9,2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

I. Introduction

It doesn't make sense for h θ (x) to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}. To fix this, let's change the form for our hypotheses h θ (x) to satisfy 0⩽h θ (x)⩽1. This is accomplished by plugging θ Tx into the Logistic Function.

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1+e^{-z}}$$

We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

Instead, our cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

We can compress our cost function's two conditional cases into one case:

$$\text{Cost}(h_\theta(x), y) = -y\ \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

We can fully write out our entire cost function as follows:

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

II. Methods and Theory

To use Gradient descent approach to minimize the loss value J(θ), We can work out the derivative part using calculus to get:

$$Repeat\ \{$$

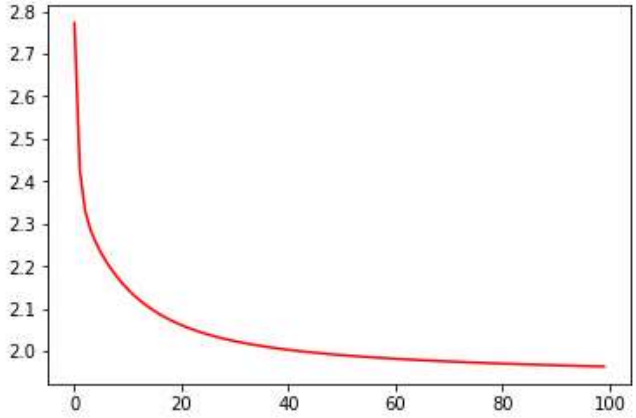
$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

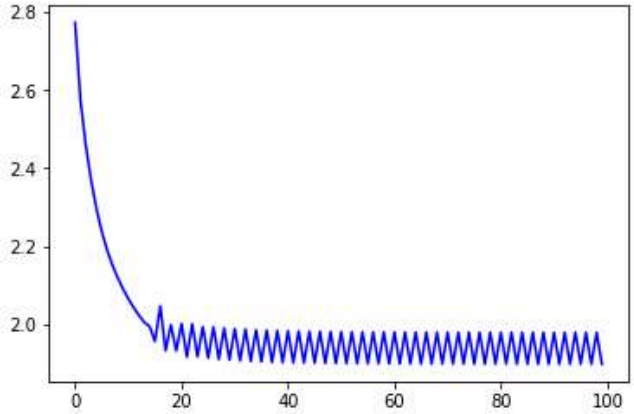
$$\}$$

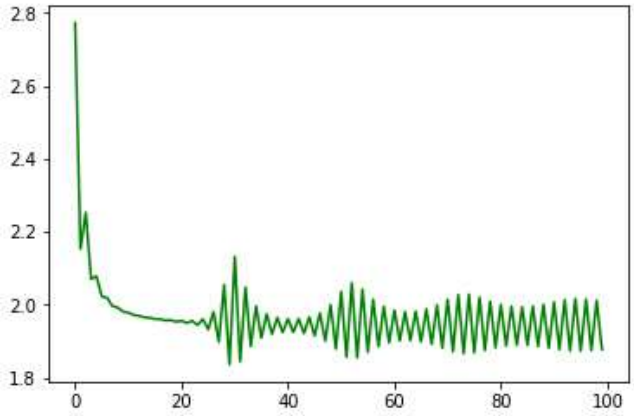
To speed up the training process,we can use some optimized gradient descent approach like AdaDelta, NAG and so on.

III. Experiment

```
def Computecost(X,y,w,m):
    J = -1/m * np.sum((1+y_train).T*np.log(sigmoid(np.dot(X_train,w))) +
                      (1-y_train).T*np.log(1-sigmoid(np.dot(X_train,w))))
    return J
def Adadelta(X,y,w,p,e,num_iters):
    m,n = np.shape(X)
    L = []
    A = []
    Eg = np.zeros(n)
    exs = np.zeros(n)
    for i in range(num_iters):
        J = Computecost(X_test,y_test,w,m_test)
        L.append(J)
        g = 1/m * np.dot((y - np.dot(X,w)).transpose(),X).T
        Eg = p*(Eg)+(1-p)*(g**2)
        delta = (np.sqrt(exs**2 + e)/np.sqrt(Eg**2 + e))*(-g)
        exs = p*exs + (1-p)*delta**2
        w = w - delta
    return L
def RMSprop(X,y,w,alpha,e,num_iters):
    m,n = np.shape(X)
    L = []
    A = []
    Eg = np.zeros(n).T
    for i in range(num_iters):
        J = Computecost(X_test,y_test,w,m_test)
        L.append(J)
        g = 1/m * np.dot((y - np.dot(X,w)).transpose(),X).T
        Eg = 0.5 * (Eg+(g**2))
        RMS = np.sqrt(Eg + e)
        w = w + alpha/RMS * g
    return L
def gradientDescent(X,y,w,alpha,num_iters):
    m,n= np.shape(X)
    L = []
    A = []
    for i in range(num_iters):
        J = Computecost(X_test,y_test,w,m_test)
        L.append(J)
        w = w + (alpha/m * np.dot((y - np.dot(X,w)).transpose(),X).T)
    return L
```







IV. CONCLUSION