0.1 Logistic Regression

0.1.1 Abstract

In this chapter, we will learn an algorithm for binary classification - soft output: Logistic regression. It mainly relies on an activation function: *sigmoid*. Since the range of the function is $(0,1)$, it can approximate the probability value.

0.1.2 Origin

The following is the explanation of sigmoid in *shuhuai* teacher's handout.

> Sometimes we only need to get the probability of a category, so we need a function that can output the value of $(0,1)$ interval. Considering the two classification model, we hope to model $p(C|x)$ using the discriminant model and Bayesian theorem:
>
> $$p\left(C_1 \mid x\right)=\frac{p\left(x \mid C_1\right) p\left(C_1\right)}{p\left(x \mid C_1\right) p\left(C_1\right)+p\left(x \mid C_2\right) p\left(C_2\right)}$$
>
> set $a=\ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$, so:
>
> $$p\left(C_1 \mid x\right)=\frac{1}{1+\exp(-a)}$$
>
> The above formula is called *Logistic Sigmoid* function and its parameters denote the logarithm of the two types of the joint probability ratios. In the discriminant, we don't care the specific value of the parameter and we just use the form of the function.

Of course, it doesn't matter if we can't understand tearcher's advanced explanation. We just need to know that we have the activation function *sigmoid* now, which can be used to get the probability of a category.

0.1.3 Algorithm

Firstly, we suppose the logistic regression model is:

$$f(x)=\sigma(w^T x)$$

In the formula, $\sigma(a)=sigmoid(a)$, we usually denote the activation function by σ.
So, if we find out the best value of w, the best model under the assumption is determined.
The parameters of probability discrinant model is usually determined by maximum likelihood estimation.
To determine the likelihood function, we have to make some marks firstly:

$$p_1=\sigma(w^T x) \qquad p_0=1-p_1$$

In the formula, p_1 is the probability of x belonging to class 1 and p_0 is the probability of class 0.
Then we can obtain the likelihood function of the model:

$$p(y|w;x) = p_1^y p_0^{1-y}$$

The likelihood function seem to be a little obscure, but actually it's reasonable:

- when y is 1: $p(y|w;x) = p_1^1 p_0^0 = p_1$
- when y is 0: $p(y|w;x) = p_1^0 p_0^1 = p_0$

Well, then we can determine the parameters via MLE.

$$\begin{aligned} \hat{w} = argmax(J(w)) &= argmax(p(Y|w;X)) \\ &= argmax(log(p(Y|w;X))) \\ &= argmax(log(\prod_{i=1}^{n} p(y_i|w;x_i))) \\ &= argmax(\sum_{i=1}^{n} log(p(y_i)|w;x_i)) \\ &= argmax(\sum_{i=1}^{n} y \; log\, p_1 + (1-y) log\, p_0) \end{aligned} \tag{1}$$

Notes, the formula is the opposite of the cross entropy formula mutiplied by N and the logarithm in MLE also match the exponential function to obtain the stable gradient in a large interval.

By differentiating the above formula, we note that:

$$p_1' = p_1(1-p_1)$$

Of course, it's easy to obtain since it's just the chain rule. We only need to be a little bit careful.
Finally, we obtain the result:

$$\frac{\partial}{\partial w} J(w) = \sum_{i=1}^{N} (y_i - p_1)\, x_i$$

Last but not least, we are to obtain the maximun of $p(p|w;x)$, so we need to use gradient ascent instead of gradient descent. Certainly they are similar, just add a negative character.

0.1.4 Implement

```
import os
os.chdir("../")
from models.linear_models import Logistic_regression
import numpy as np
import warnings

epsilon = 1
num_test = 100
num_base = 1000
ratio = 0.6
k1, k2 = 3, 5
b1, b2 = 1, 2
X = np.linspace(0, 100, num_base)
X_train = X[:-num_test]
X_test = X[-num_test:]
v1 = X_train[:round(len(X_train) * ratio)] * k1 + b1
v2 = X_train[round(len(X_train) * ratio):] * k2 + b2
v1 += np.random.normal(scale=epsilon, size=v1.shape)
v2 += np.random.normal(scale=epsilon, size=v2.shape)
value = np.r_[v1, v2]
data = np.c_[X_train, value]
l1 = np.ones_like(v1)
l2 = np.zeros_like(v2)
label = np.r_[l1, l2]
v_test_c1 = X_test * k1 + b1
l_test_c1 = np.ones_like(v_test_c1)
data_test = np.c_[X_test, v_test_c1]

model = Logistic_regression(10, 1000, lr=1e-3)
model.fit(data, label)
print(model.get_params())
print(model.predict(data_test, l_test_c1))
```