机器学习导论

姓名：殷天润　　**学号：**171240565

2019 年 4 月 1 日

请独立完成作业，不得抄袭。
若参考了其它资料，请给出引用。
鼓励讨论，但需独立书写解题过程。

第一部分　作业

☑ Problem 1 (ML problem 1)

In multi-label problem, each instance $\boldsymbol{x}$ has a label set $\boldsymbol{y} = \{y_1, y_2, ..., y_l\}$ and each label $y_i \in \{0, 1\}$. Assume the post probability $p(\boldsymbol{y}|x)$ follows the conditional independence:

$$p(\boldsymbol{y}|x) = \prod_{i=1}^{l} p(y_i|x). \tag{1}$$

Please use the logistic regression method to handle the following questions.
(1) [15pts] Please give the 'log-likelihood' function of your logistic regression model;
(2) [10pts] Please calculate the gradient of your 'log-likelihood' function.

Solution

1. 根据y_i的组合,$Y_i = (y_1, ...y_n), y_i = \{0, 1\}$有$2^l$次方种可能性;
   由于条件独立,对于每一种Y_i都有$p(Y_i) = \Pi_{j=1}^{l} p(y_{ij}|x)$;

$$\begin{aligned} & l(\boldsymbol{w}_1, \boldsymbol{w}_2, ...\boldsymbol{w}_l; b_1, b_2, ..., b_l) \\ &= \sum_{i=1}^{m} ln\, p(Y_i|\boldsymbol{x}; \boldsymbol{w_i}, b_i) \\ &= \sum_{i=1}^{m} ln\, (\Pi_{j=1}^{l} p(y_{ij}|x; \boldsymbol{w_i}, b_i)) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{l} ln(p(y_{ij}|x; \boldsymbol{w_i}, b_i)) \end{aligned} \tag{2}$$

   为了简化书写,增广$\boldsymbol{x}$,令$\hat{\boldsymbol{x}} = (\boldsymbol{x}, 1)$,$\boldsymbol{\beta}_i = (\boldsymbol{w}_i, b_i)$,此时可以简写$\boldsymbol{w}^T\boldsymbol{x} + b$为$\boldsymbol{\beta}_i^T\boldsymbol{x}$;
   模仿西瓜书p59页的处理:令$p_{j1}(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}_i) = p(y_{ij} = 1|\hat{\boldsymbol{x}}; \boldsymbol{\beta}_i)$(表示第i组第j个y标签为1的概率)，$p_{j0}(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}_i) = p(y_{ij} = 0|\hat{\boldsymbol{x}}; \boldsymbol{\beta}_i) = 1 - p(y_{ij} = 1|\hat{\boldsymbol{x}}; \boldsymbol{\beta}_i)$
   因此可以重写极大似然项为:

$$p(y_{ij}|\boldsymbol{x}_i;\boldsymbol{w}_i,b_i) = y_i p_{j1}(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}_i);\boldsymbol{\beta}_i) + (1-y_i)p_{j0}(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}_i)$$

将这个式子带入到公式(2)中,将最大化(2)式转化成最下化下式有:

$$l(\boldsymbol{\beta_1},\boldsymbol{\beta_2},\boldsymbol{\beta_3}...,\boldsymbol{\beta_l}) = \sum_{i=1}^{m}\sum_{j=1}^{l}(-y_{ij}\beta_j^T\hat{x}_i + ln(1+e^{\beta_j^T\hat{x}_i}))$$

2. 梯度计算:

$$\begin{aligned}\nabla(l(\boldsymbol{\beta_1},\boldsymbol{\beta_2},\boldsymbol{\beta_3}...,\boldsymbol{\beta_l})) &= \frac{\partial l(\boldsymbol{\beta_1},\boldsymbol{\beta_2},\boldsymbol{\beta_3}...,\boldsymbol{\beta_l})}{\partial(\boldsymbol{\beta_1},\boldsymbol{\beta_2},\boldsymbol{\beta_3}...,\boldsymbol{\beta_l})} \\ &= -\sum_{i=1}^{m}\sum_{j=1}^{l}\hat{\boldsymbol{x}}_i(y_{ij} - p_{j1}(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}_j))\end{aligned}$$

Problem 2 (ML problem 2)

Suppose we transform the original $\mathbf{X}$ to $\hat{\mathbf{Y}}$ via linear regression . In detail, let

$$\hat{\mathbf{Y}} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-\mathbf{1}}\mathbf{X}^\top\mathbf{Y} = \mathbf{X}\hat{\mathbf{B}},$$

where $\mathbf{X}$ and $\mathbf{Y}$ are the feature and label matrix, respectively. Similarly for any input $\mathbf{x}$, we get a transformed vector $\hat{\mathbf{y}} = \hat{\mathbf{B}}^\top\mathbf{x}$. Show that LDA using $\hat{\mathbf{Y}}$ is identical to LDA in the original space.

Solution

对于给定的数据集D=$\{(x_i,y_i)\}_{i=1}^m, y_i = \{0,1\}$,定义class数量是K

1. 首先讨论X的平均值,便于后面替换:

$$\mu = \frac{1}{m}\sum_{i=1}^{m}x$$

经过变换后:

$$\hat{\mu} = \frac{1}{m}\sum_{i=1}^{m}\hat{y} = \frac{1}{m}\hat{B}^T\sum_{i=1}^{m}x = \hat{B}^T\mu$$

2. 现在讨论类内散度矩阵(within-class scatter matrix)和类间散度矩阵(scatter matrix)

$$S_w = \sum_{i=1}^{K}\sum_{x\in X_i}(x-\mu_i)(x-\mu_i)^T$$

$$\hat{S_w} = \sum_{i=1}^{K}\sum_{x\in X_i}(\hat{B}^Tx - \hat{B}^T\mu_i)(\hat{B}^Tx - \hat{B}^T\mu_i)^T = \hat{B}^T\sum_{i=1}^{K}\sum_{x\in X_i}(x-\mu_i)(x-\hat{\mu}_i)^T\hat{B} = \hat{B}^TS_w\hat{B}$$

同理,

$$\hat{S_b} = \hat{B}^TS_b\hat{B}$$

3. 因此现在考虑J(w)

$$J(w) = \frac{w^TS_bw}{w^TS_ww}$$

$$\hat{J(w)} = \frac{w^T\hat{S_b}w}{w^T\hat{S_w}w} = \frac{w^T\hat{B}^TS_b\hat{B}w}{w^T\hat{B}^TS_w\hat{B}w} = \frac{(\hat{B}w)^TS_b(\hat{B}w)}{(\hat{B}w)^TS_w(\hat{B}w)}$$

令w'=$\hat{B}w$,可见LDA using $\hat{\mathbf{Y}}$ is identical to LDA in the original space.

Problem 3 (ML problem 3)

0.1 [55pts] Logistic Regression from scratch

Implementing algorithms is a good way of understanding how they work in-depth. In case that you are not familiar with the pipeline of building a machine learning model, this article can be an example (here).

In this experiment, you are asked to build a classification model on one of UCI data sets, Letter Recognition Data Set (click to download). In particular, the objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The detailed statistics of this data set is listed in Table 1. The data set was then randomly split into train set and test set with proportion $7 : 3$. Also, letters from 'A' to 'Z' are mapped to digits '1' to '26' respectively as represented in the last column of the provided data set.

Table 1: Statistics of the data set.

Property	Value	Description
Number of Instances	20,000	Rows of the data set
Number of Features	17	Columns of the data set
Number of classes	26	Dimension of the target attribute

In order to build machine learning models, you are supposed to implement Logistic Regression (LR) algorithm which is commonly used in classification tasks. Specifically, in this experiment, you have to adapt the traditional binary class LR method to tackle the multi-class learning problem.

(1) [**5pts**] You are encouraged to implement the code using *Python3* or *Matlab*, implementations in any other programming language will not be judged. Please name the source file (which contains the main function) as *LR_main.py* (for python3) or *LR_main.m* (for matlab). Finally, your code needs to print the testing performance on the provided test set once executed.

(2) [**30pts**] Functions required to implement:
- Implement LR algorithm using gradient descent or Newton's method.
- Incorporate One-vs-Rest (OvR) strategy to tackle multi-class classification problem.

(3) [**20pts**] Explain implementing details in your submitted report (source code should not be included in your report), including optimization details and hyper-parameter settings, etc. Also, testing performance with respect to Accuracy, Precision, Recall, and F_1 score should be reported following the form of Table 2.

NOTE: Any off-the-shelf implementations of LR or optimization methods are **NOT ALLOWED** to use. When submitting your code and report, all files should be placed in the same directory (without any sub-directory).

Solution

1. 我已经独立写了python程序,名字是LR_main.py;默认情况下:步长为0.09,迭代次数1000次,大约需要分钟时间运行;运行前首先要确保环境是python3,需要有numpy,pandas以及csv包;
2. 我使用了梯度下降法以及OVR来实现这个程序,梯度下降法实现了class LogisticClassfier,OVR思想实现了class OVRclassifier;我在之前尝试的时候还试过了随机梯度下降算法,但是效果不是很好,具体细节下面会说到;
3. 这部分我将详细的描述我的实验,主要分为:程序模块介绍,程序优化过程,调参数据,实验结果这几个部分

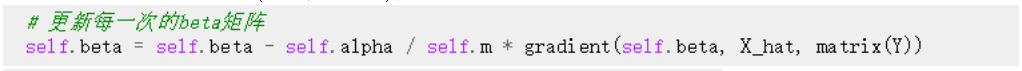
Table 2: Performance of your implementation on test set.

Performance Metric	Value (%)
accuracy	00.00
micro Precision	00.00
micro Recall	00.00
micro F_1	00.00
macro Precision	00.00
macro Recall	00.00
macro F_1	00.00

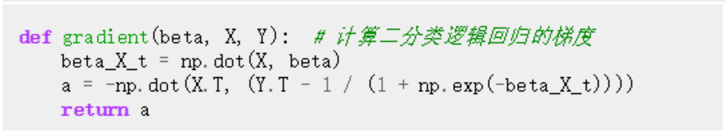
(a) 程序各个模块的介绍: 我的程序主要包括:两个class:class LogisticClassfier,class OVRclassifier,四个def: def gradient,def sigmod_fit_matrix,def sigmod_output_matrix,def score组成;

class LogisticClassfier:用梯度下降法实现的二分类器:

i. 训练的模块是def fit(self, X, Y),核心的部分是:



```
# 更新每一次的beta矩阵
self.beta = self.beta - self.alpha / self.m * gradient(self.beta, X_hat, matrix(Y))
```



```
def gradient(beta, X, Y):  # 计算二分类逻辑回归的梯度
    beta_X_t = np.dot(X, beta)
    a = -np.dot(X.T, (Y.T - 1 / (1 + np.exp(-beta_X_t))))
    return a
```

ii. 预测的部分是def predict(self, X),用训练出来的beta通过sigmod函数进行预测

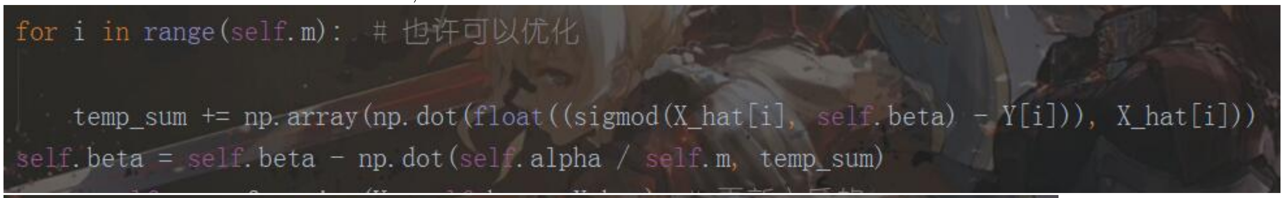
class OVRclassifier:多分类器

i. def OVR_train(self, X_train, Y_train):
每一次选一个(chosen_one)作为 "O",在Y_train将对应的取值标为1,其余标为0,送给相应的二分类器list中的二分类器训练即可;

ii. def OVR_predict(self, X_test):
运用相应的二分类器中的predict进行预测;

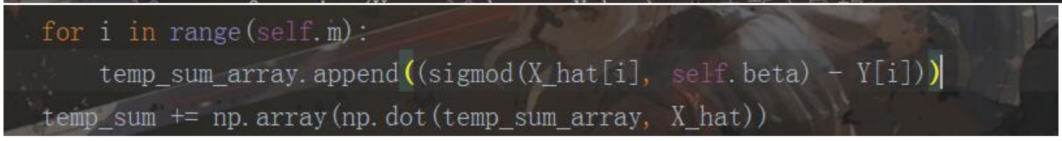
(b) 程序的优化过程:

i. 我主要对程序的速度进行了优化;



```
for i in range(self.m):  # 也许可以优化

    temp_sum += np.array(np.dot(float((sigmod(X_hat[i], self.beta) - Y[i])), X_hat[i]))
self.beta = self.beta - np.dot(self.alpha / self.m, temp_sum)
```

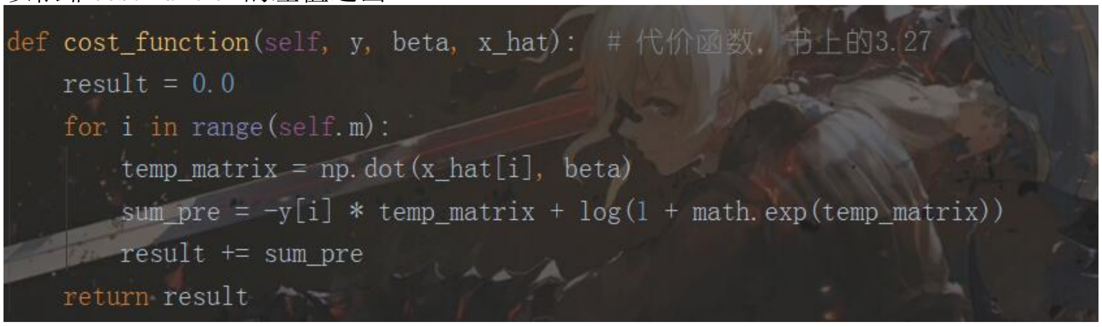


```
for i in range(self.m):
    temp_sum_array.append((sigmod(X_hat[i], self.beta) - Y[i]))
temp_sum += np.array(np.dot(temp_sum_array, X_hat))
```

上面是我之前两个版本的程序,第一个版本完全是按照西瓜书上的3.30式子写的,是一个for循环的矩阵运算;这个版本速度实在太慢;因此我修改成了下一个版本,减少了矩阵的运算,瓶颈变为了for循环下的sigmod函数运算,由于这个是对单个的元素进行运行,速度虽然是上个版本的感觉上的两倍,但是还是很慢,算一千次大概需要2-3个小时;因此我改成了目前这个版本,删去了for循环,将它化简为仅仅一个矩阵运算,速度快了很多,1000次的loop只需要5-10分钟!

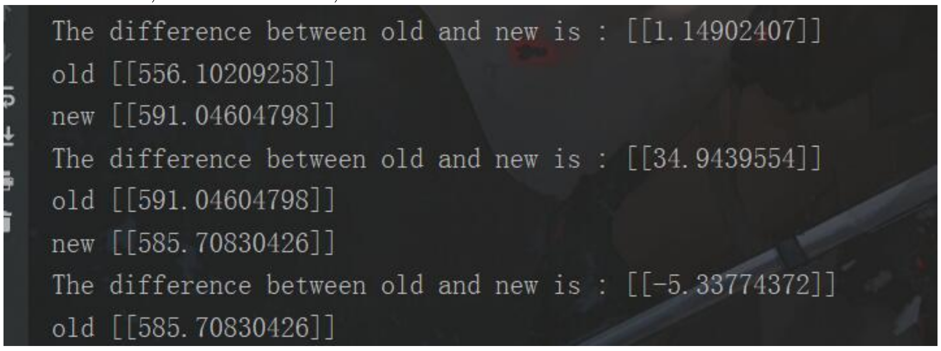
另外我选择了取平均值操作也就是除以m,这样对步长的要求更小而且每一次更新更加平稳;

ii. 除了仅仅依靠loop次数的退出方式,在调参的过程中我还根据书上的3.27式子,使用了辅助的cost_function,可以依靠cost_funtion的差值退出:



```
def cost_function(self, y, beta, x_hat):  # 代价函数, 书上的3.27
    result = 0.0
    for i in range(self.m):
        temp_matrix = np.dot(x_hat[i], beta)
        sum_pre = -y[i] * temp_matrix + log(1 + math.exp(temp_matrix))
        result += sum_pre
    return result
```

在运行梯度下降前我先跑一遍cost_function,与运行后的值进行比较,就可以知道梯度下降对于beta拟合的贡献度,如果步长正常,前后两次的cost_function的值应该差一个小的负数;如果出现



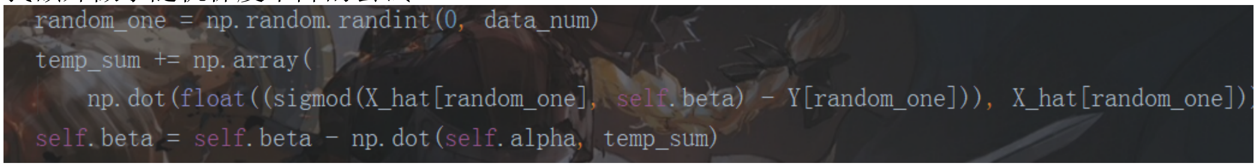
```
The difference between old and new is : [[1.14902407]]
old [[556.10209258]]
new [[591.04604798]]
The difference between old and new is : [[34.9439554]]
old [[591.04604798]]
new [[585.70830426]]
The difference between old and new is : [[-5.33774372]]
old [[585.70830426]]
```

这种情况,那么就可以一定意义上的判定自己的步长取的偏大,在不需要算出最后精确度的情况下就可以修改,一定意义上减少了调参时间;

另外我在调参时也设计运行到loop的$\frac{9}{10}$之后就开始计算cost_function的差值,并且设置accuracy,让差值小于这个accuracy之后就直接退出,避免过度拟合

(c) 调参:

i. 我额外做了随机梯度下降的尝试:



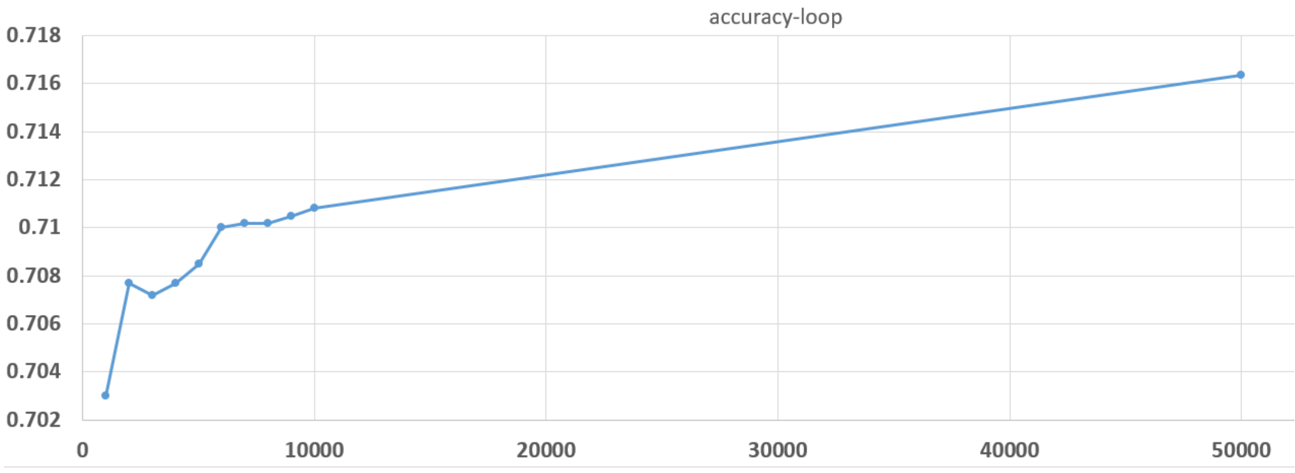
```
random_one = np.random.randint(0, data_num)
temp_sum += np.array(
    np.dot(float((sigmod(X_hat[random_one], self.beta) - Y[random_one])), X_hat[random_one])
self.beta = self.beta - np.dot(self.alpha, temp_sum)
```

但是效果并不好,当alpha取9.6e-07的时候,loop为500000的时候准确率为25.8%,当alpha取0.0001的时候,loop
时候准确率为13.25%

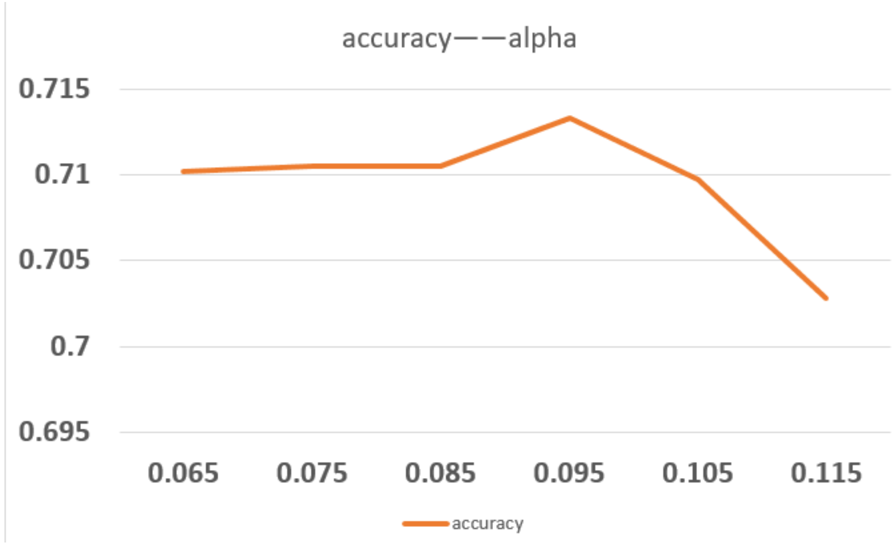
ii. 我控制loop以及控制alpha做了相应的对比实验:

可视化图表如下:

A. 固定alpha为0.09,改变loop次数;纵坐标的值表示accuracy,横坐标是loop次数:

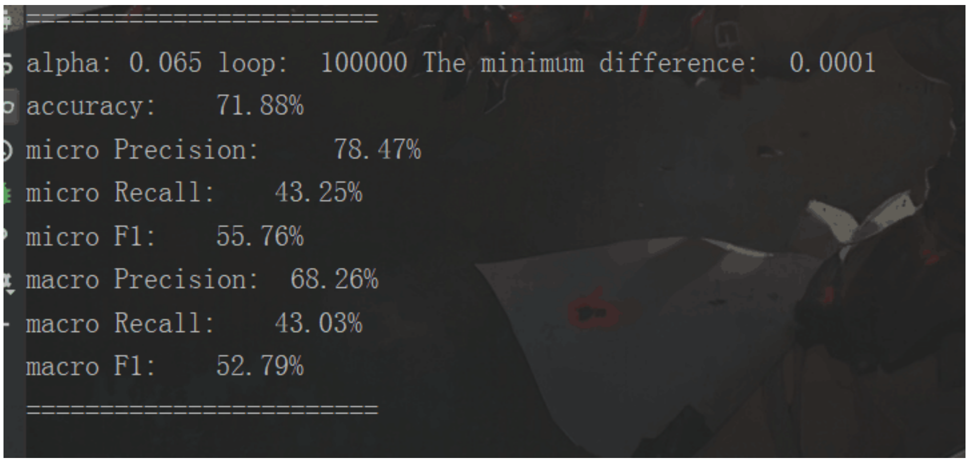


B. 固定loop次数为10000,改变alpha;纵坐标的值表示accuracy,横坐标是alpha:



(d) 实验结果:

我选择下图作为我实验的结果,跑了十万次loop大概需要2-3个小时,如果复现请选择0.09,1000次这样的默认参数作为输入;

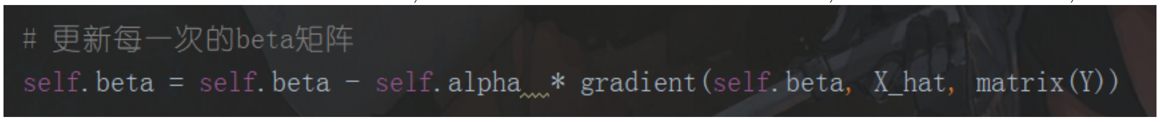


```
========================
alpha: 0.065 loop:  100000 The minimum difference:  0.0001
accuracy:    71.88%
micro Precision:     78.47%
micro Recall:    43.25%
micro F1:    55.76%
macro Precision:   68.26%
macro Recall:    43.03%
macro F1:    52.79%
========================
```

(e)

4. 其他内容:

- 除了上面所说的随机梯度下降法,我还尝试了修改梯度下降的式子,将平均值改成总数,



```
# 更新每一次的beta矩阵
self.beta = self.beta - self.alpha * gradient(self.beta, X_hat, matrix(Y))
```

Table 3: Performance of my implementation on test set.

Performance Metric	Value (%)
accuracy	71.88
micro Precision	78.47
micro Recall	43.25
micro F_1	55.76
macro Precision	68.26
macro Recall	43.03
macro F_1	52.79

这种情况对alpha的精度要求比较高,需要更多的loop才能获得同等的结果(70%左右),并且观察cost_function的dif
时候我发现波动比较大,常常有正值;

并且这种方法需要比较多的loop才能获得相对准确的结果;因此我还是使用之前平均值的用于梯度的更新

- 观察到这次的数据数量级相近,因此不对数据进行归一化处理