中山大学计算机学院
人工智能
本科生实验报告

课程名称：Artificial Intelligence

学号	22336327	姓名	庄云皓

一、 实验题目

购房预测分类任务

二、 实验内容

1. 算法原理

1.1 逻辑回归：

对于是否买房这个二分类问题，给定数据集

$$D = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N), x_i \subseteq R^n, y_i \in 0, 1, i = 1, 2, \cdots, N$$

N 个样例，每个样例 n 个特征

xi 为每个样本的所有特征;

y 是 label 列。

为了方便，我们将输入向量进行**扩充**

$$x \ = \left(x^{(1)}, x^{(2)}, \dots, x^{(n)}, 1\right)$$

$$\mathrm{w} = \left(\mathrm{w}^{(1)}, \mathrm{w}^{(2)}, \dots, \mathrm{w}^{(n)}, \mathrm{b}\right)$$

在是否买房我们用线性函数 h(x)= $w_1x_1 + w_2x_2 + b = 0$，来拟合一个决策边界

如果某点的 h(x)>0,则判断类别为 1，反之为 0.

因为 h(x)是连续的，不能拟合一个离散变量（分类结果是 0/1），所以我们用它来拟合条概率 p(Y = 1|x),也就是在给定输入向量 xi 的条件 yi = 1 的概率(因为概率的取值可以是连续的)。

一个事件发生的概率 p,则不发生的概率为 1-p，用机率（odds）来表示这两者的比值,再取对数得到对数几率（log odds)d：

$$\mathrm{logit}(p) = \ln\frac{p}{1-p}$$

对 logistic 回归来说，令 P = P(Y=1|x),得

$$\ln\frac{P(Y=1|x)}{1-P(Y=1|x)} = w\cdot x$$

这就是说，在 logistic 回归模型中，给定 x 输出 Y =1 的对数几率是输入 x 的线性函数
解上面的方程得到如下结果，换一个角度看，这实际上是将线性函数转化为概率：

$$P(Y=1|x)=\frac{\exp(w\cdot x)}{1+\exp(w\cdot x)}$$

同样地：

$$P(Y=0|x)=\frac{1}{1+\exp(w\cdot x)}$$

这就是 logistic 回归模型。

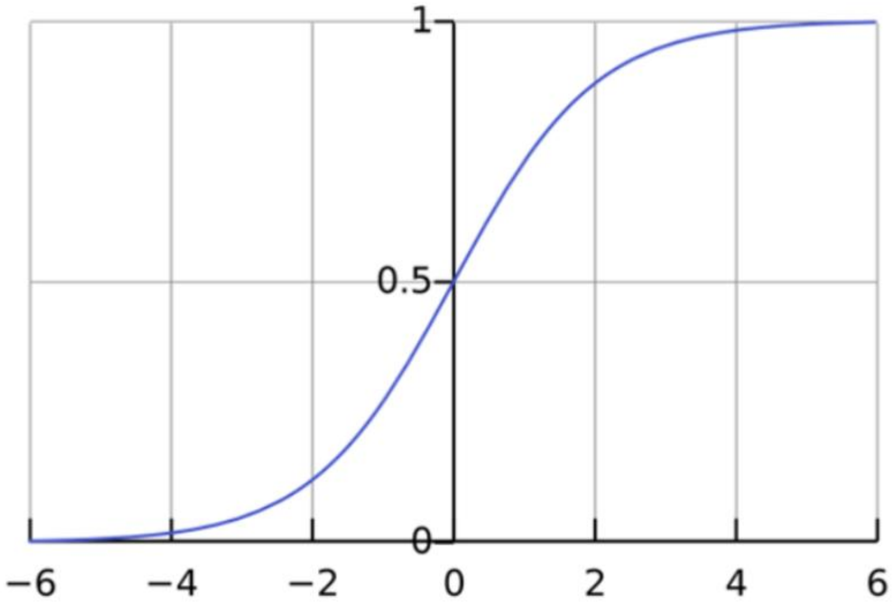
可以看出，这是一个 sigmoid 函数：

$$f(x)=\frac{1}{1+e^{-x}}$$

把 h(x)作为自变量时的情形即为：

$$y=\frac{1}{1+e^{-(w^Tx+b)}}$$

Sigmoid 函数图象如图所示



模型参数估计：用极大似然估计法求模型参数

令π(x) = P(Y = 1|x)

对数似然函数

$$\begin{aligned} L(w) &= \sum_{i=1}^{N}[y_i \log \pi(x_i) + (1-y_i)\log(1-\pi(x_i))] \\ &= \sum_{i=1}^{N}\left[y_i \log\frac{\pi(x_i)}{1-\pi(x_i)} + \log(1-\pi(x_i))\right] \\ &= \sum_{i=1}^{N}[y_i(w \cdot x_i) - \log(1+\exp(w \cdot x_i))] \end{aligned}$$

求 L(w)的极大值点即为 w 估计值$\bar{w}$,我们采用梯度上升法求极大值。

这等同是交叉熵函数J(θ)= -L(ω)，采用梯度下降法求极小值

（直接放公式图片了，这里α为学习率）

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}\sum_{i=1}^{m}(y^i - \hat{y^i})X$$

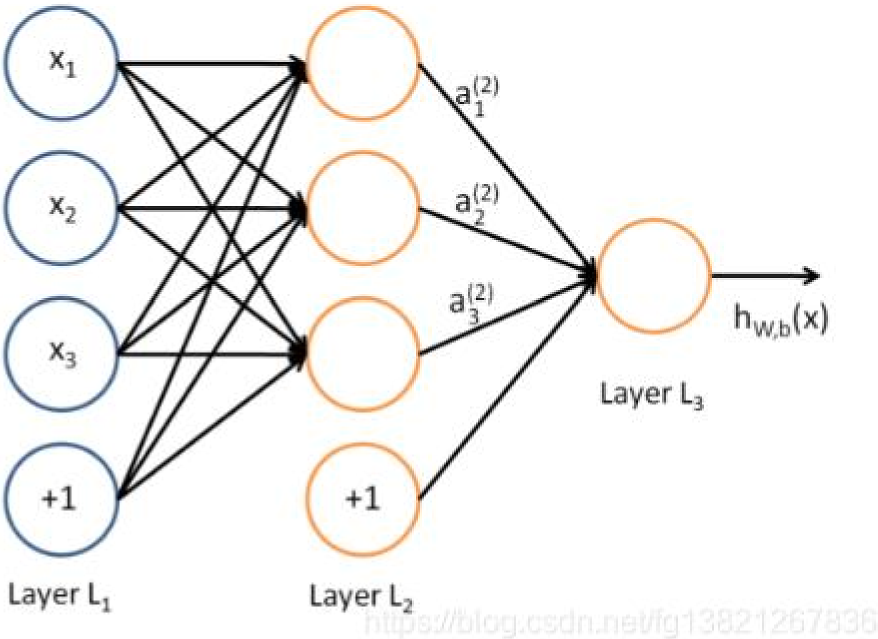
$$\theta = \theta - \alpha\frac{\partial J(\theta)}{\partial \theta}$$

代码表示即

```
grad = X.T.dot(y_hat - y)/len(y)
W-=lr*grad
```

1.2 感知机

多层感知机（MLP，Multilayer Perceptron）也叫人工神经网络（ANN，Artificial Neural Network），除了输入输出层，它中间可以有多个隐层，最简单的 MLP 只含一个隐层，即三层的结构，如下图：



具体来说，代码对应的 MLP 模型中采用了一个隐藏层神经元数量为 2，隐藏层和输出层都通过 sigmoid 函数激活的神经网络。

Forward 部分：

output_hidden = sigmoid(W1X+b1)

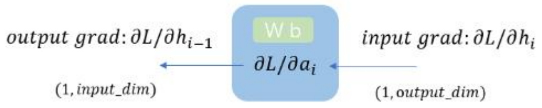
y_hat = sigmoid(W2X+b2)

损失函数

Loss = mean(-sum(y*log(y_hat)-(1-y)*log(1-y_hat)))

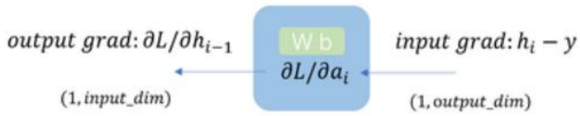
反向传播：

注意代码和公式中 * 表示element-wise乘积，·表示矩阵乘积。



$$\frac{\partial L}{\partial a_i}=\frac{\partial L}{\partial h_i}*activate(a_i)\qquad \frac{\partial L}{\partial b_i}=\frac{\partial L}{\partial a_i}$$

$$\frac{\partial L}{\partial h_{i-1}}=\frac{\partial L}{\partial a_i}\cdot W^T\qquad \frac{\partial L}{\partial W_i}=h_i^T\cdot\frac{\partial L}{\partial a_i}$$

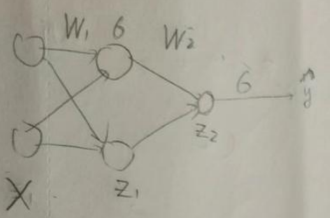


$$\partial L/\partial a_i=h_i-y\qquad \frac{\partial L}{\partial b_i}=\frac{\partial L}{\partial a_i}$$

$$\frac{\partial L}{\partial h_{i-1}}=\frac{\partial L}{\partial a_i}\cdot W^T\qquad \frac{\partial L}{\partial W_i}=h_i^T\cdot\frac{\partial L}{\partial a_i}$$

求梯度+更新权重：

手推了一下公式，不一定对



前向

$$z_1=X\cdot W_1^T+b$$

$$h_1=\sigma(z_1)$$

$$z_2=h_1W_2^T+b$$

$$\hat{y}=h_2=\sigma(z_2)$$

$$L=-np.mean\left(y\ln\hat{y}+(1-y)\ln(1-\hat{y})\right)$$

如果L使用的是L2范数而非交叉熵则末尾要乘上 $\hat{y}(1-\hat{y})$

反向

$$\frac{\partial L}{\partial h_2}=\frac{\partial L}{\partial \hat{y}}=-y\frac{1}{\hat{y}}-(1-y)\frac{-1}{1-\hat{y}}$$

$$\frac{\partial L}{\partial z_2}=\frac{\partial h_2}{\partial z_2}\frac{\partial L}{\partial h_2}=\sigma'(z_2)\cdot(\quad)$$

$$=(\quad)\sigma(z_2)(1-\sigma(z_2))$$

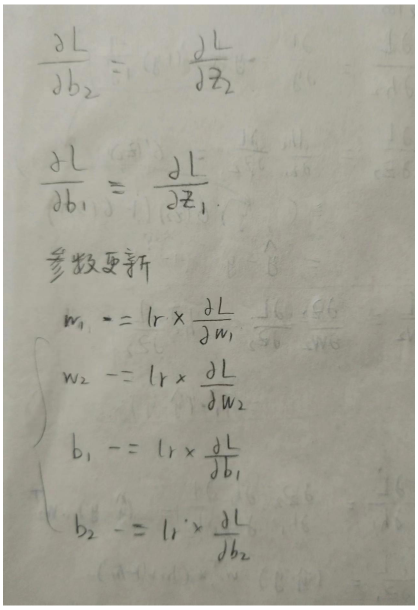
$$=\hat{y}-y$$

$$\boxed{\frac{\partial L}{\partial W_2}=\frac{\partial z_2}{\partial W_2}\frac{\partial L}{\partial z_2}=h_1^T\frac{\partial L}{\partial z_2}=h_1^T\cdot(\hat{y}-y)}$$

$$\frac{\partial L}{\partial h_1}=\frac{\partial z_2}{\partial h_1}\cdot\frac{\partial h_2}{\partial z_2}\cdot\frac{\partial L}{\partial h_2}=(\hat{y}-y)\cdot W_2^T$$

$$\frac{\partial L}{\partial z_1}=(\hat{y}-y)\cdot W_2^T*(h_1*(1-h_1))$$

$$\boxed{\frac{\partial L}{\partial W_1}=\frac{\partial z_1}{\partial W_1}\cdot\frac{\partial L}{\partial z_1}=X^T\cdot\frac{\partial L}{\partial z_1}}$$



求偏导的代码见关键代码部分

不难看出，partial(L)/partial(w1)其偏导值在输出值 y_hat 接近 0 或者 1 的时候都会非常小，这可能会造成模型刚开始训练时，偏导值几乎消失，导致 w1 权重几乎不发生变化。

2. 关键代码展示

2.1 逻辑回归：

```
def train(self):
    losses = []
    X, y = read_csv("data/data.csv")
    X[:, 0] = (X[:, 0] - np.mean(X[:, 0]))/ np.std(X[:, 0]) # 第 0 列均值方差归一化
    X[:, 1] = (X[:, 1] - np.mean(X[:, 1]))/ np.std(X[:, 1])  # 第 1 列均值方差归一化
    w = np.random.randn(X.shape[1])#从标准正态分布中随机取值，作为初始值
    y_hat = self.model(X,w,y)
    for i in range(self.max_iter):
        grad = X[:].T.dot(y_hat - y)/len(y)
      #   w[2] = -np.mean(y-y_hat+w[2])
        #grad = np.concatenate((grad,w[2]), axis=0)
        np.append(grad,w[2])
        w = w - self.lr*grad # 更新权重
        y_hat = self.model(X,w,y) #计算出 y_hat
        losses.append(loss(y_hat,y)) #loss append 到 losses 中
        if (abs(loss(y_hat,y))<1e-6):
            break;
    plt.plot(losses)
```

```
        plt.show()
        self.plotBestFit(w)
```

2.2MLP:

求梯度

```
      #求对层输出的梯度
       L2_delta=(output-self.out)

       # print(np.shape(L2_delta))
       L1_delta = L2_delta.dot(self.w2.T) * my_mlp.d_sigmoid(self.h_out)
       #求对参数的梯度
       d_w2 = rate * self.h_out.T.dot(L2_delta)
       d_w1 = rate * input.T.dot(L1_delta)
```

更新权重：

```
       self.b2 += rate*d_b2.reshape(d_b2.shape[0]*d_b2.shape[1],)
       self.b1 += rate*d_b1.reshape(d_b1.shape[0]*d_b1.shape[1],)
       d_b2 = np.ones((1,sample_num)).dot(L2_delta)
       d_b1 = np.ones((1,sample_num)).dot(L1_delta)
```

3. 创新点&优化

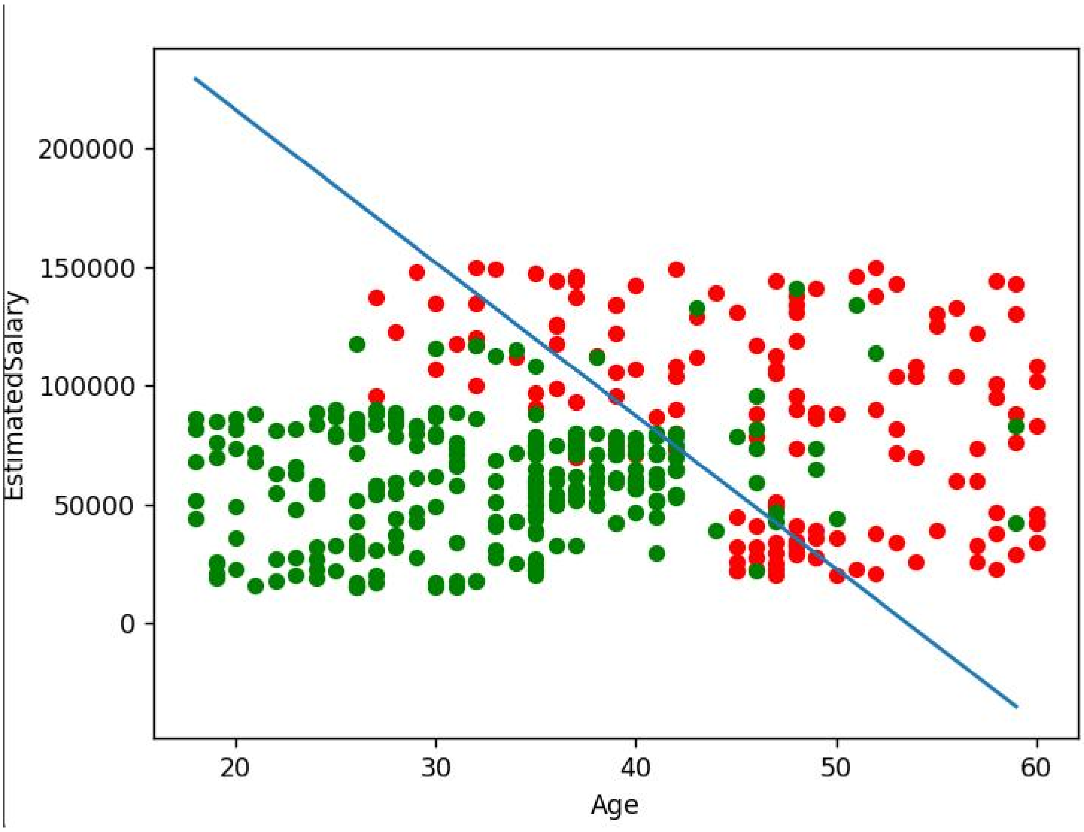
由于 BGD 计算量较大，在 MLP 中的优化中采用了小批量梯度下降的方法，每次选取一定数目(mini-batch)的样本组成一个小批量样本，然后用这个小批量来更新梯度，这样不仅可以减少计算成本，还可以提高算法稳定性。

```
      indices = np.random.choice(data_size, batch_size)
       x_batch = x_data[indices]
       y_batch = y_data[indices]
       mlp.backpropagation(x_batch,y_batch) #反向传播
       out=mlp.forward(x_data)
       loss = -np.mean(y_data*np.log(out)+(1-y_data)*np.log(1-out))
```

三、 实验结果及分析

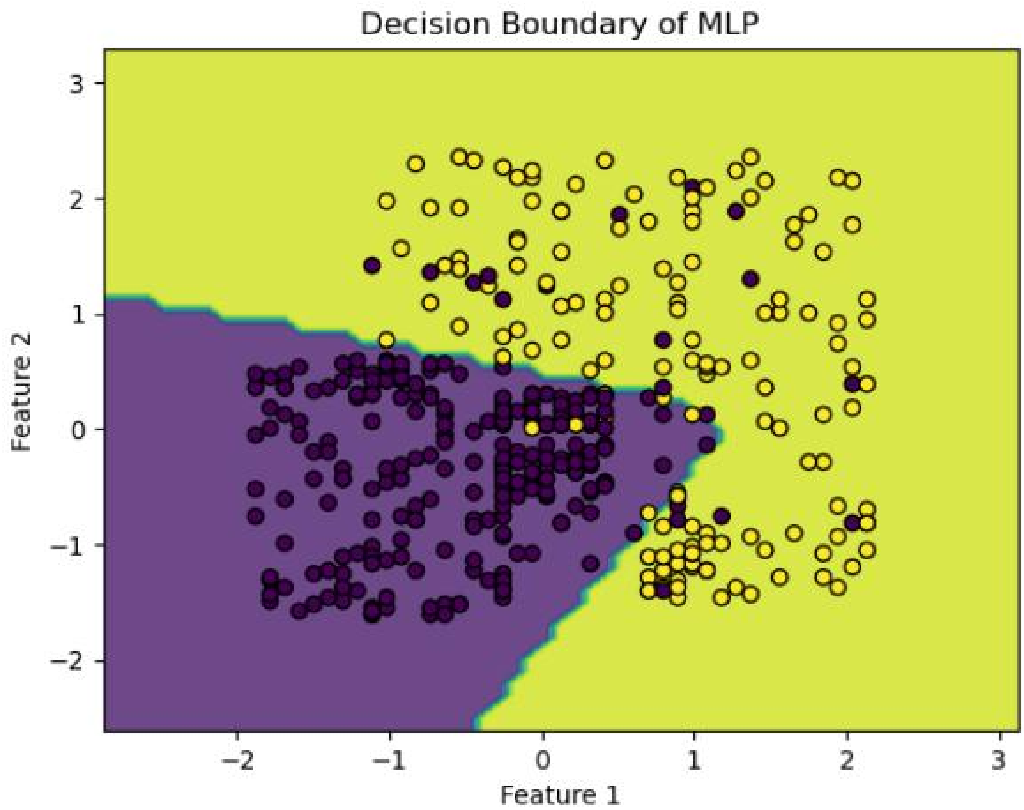
1. 实验结果展示示例

逻辑回归：



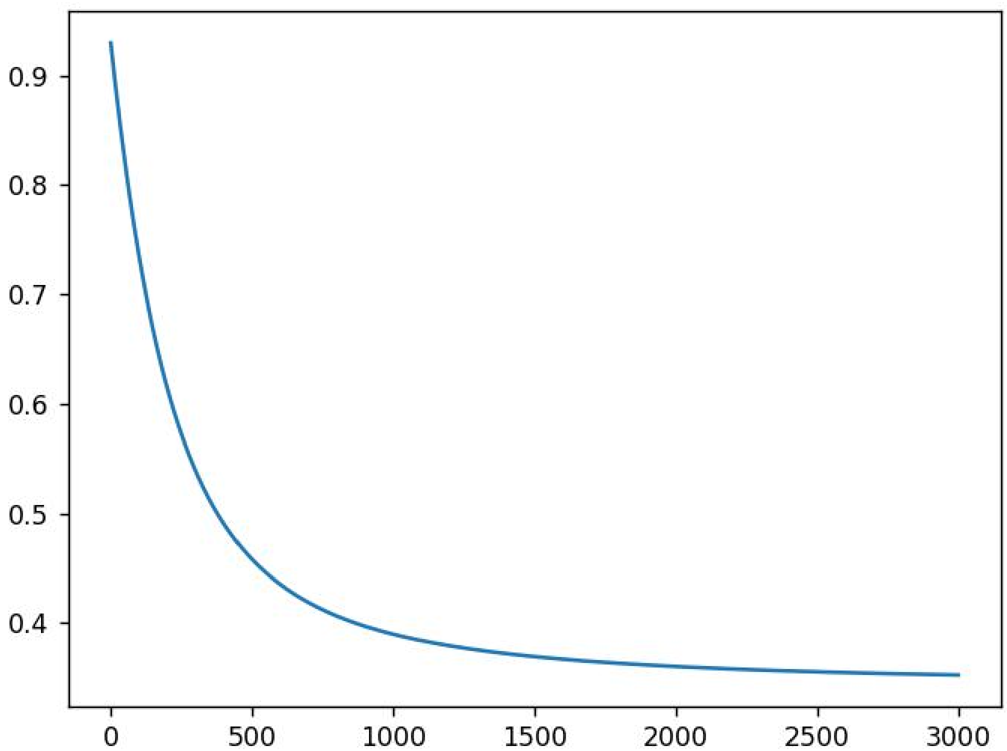

```
acc: 0.84
```

MLP(n_hidden = 2):
横纵坐标值是归一化之后的特征的值

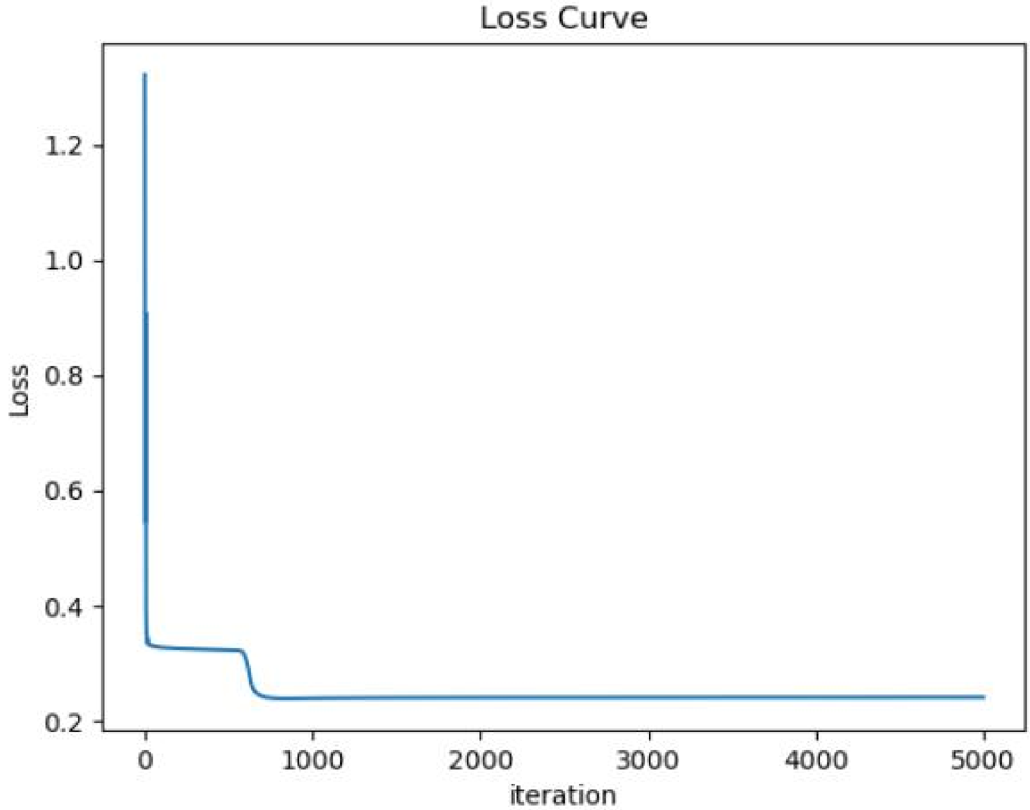


2. 评测指标展示及分析

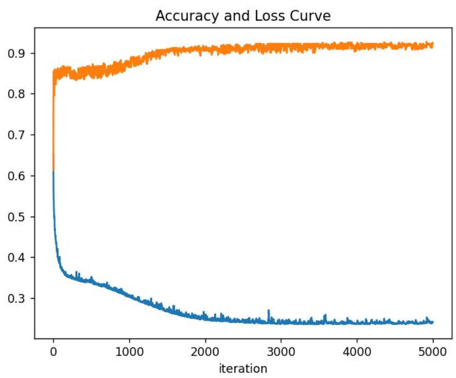
逻辑回归：



MLP(n_hidden = 2):



采用小批量梯度下降取代每次用所有数据进行梯度下降进行优化，使得每次的计算量减少了，发现收敛速度更加均匀，没有出现 Loss 突然下降的情况。但是不能保证找到最优的决策边界。




```
交叉熵损失函数的值: 0.24079644316940993
准确率: 0.925
```

遇到的一个问题：

#RuntimeWarning: overflow encountered in exp

  return 1. / (1. + np.exp(-x))#np.exp 函数

要先进行数据标准化与归一化防止 1. / (1. + np.exp(-x))越界

四、 参考资料

LogisticRegression 逻辑回归(附代码实现) - 知乎 (zhihu.com)
【Numpy】中 np.random.rand()和 np.random.randn()的用法和区别_python np.random.randn-CSDN 博客
sw_machine_learning/machine_learning_algorithm/logistic_regression at master · yunshuipiao/sw_machine_learning (github.com)
《统计学习方法》 李航
深度学习 | 反向传播详解 - 知乎 (zhihu.com)
MLP 多层感知机用 BP 算法更新权值解决异或问题(机器学习实验二)_基于 mlp 解决异或问题-CSDN 博客