**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** |

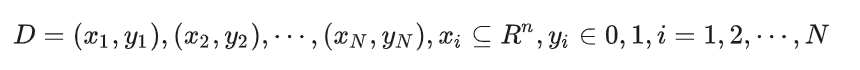
# 实验题目

**购房预测分类任务**

# 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

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**

在是否买房我们用线性函数h(x)=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.

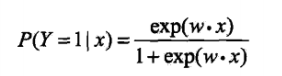
因为h(x)是连续的，不能拟合一个离散变量（分类结果是0/1）, 所以我们用它来拟合条概率p(Y = 1|x),也就是在给定输入向量xi的条件yi = 1的概率（因为概率的取值可以是连续的）。

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：

对logistic 回归来说，令P = P(Y=1|x),得

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：

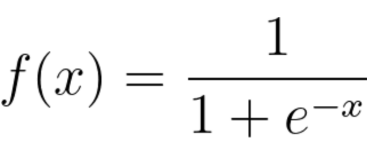


同样地：

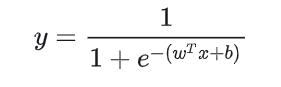


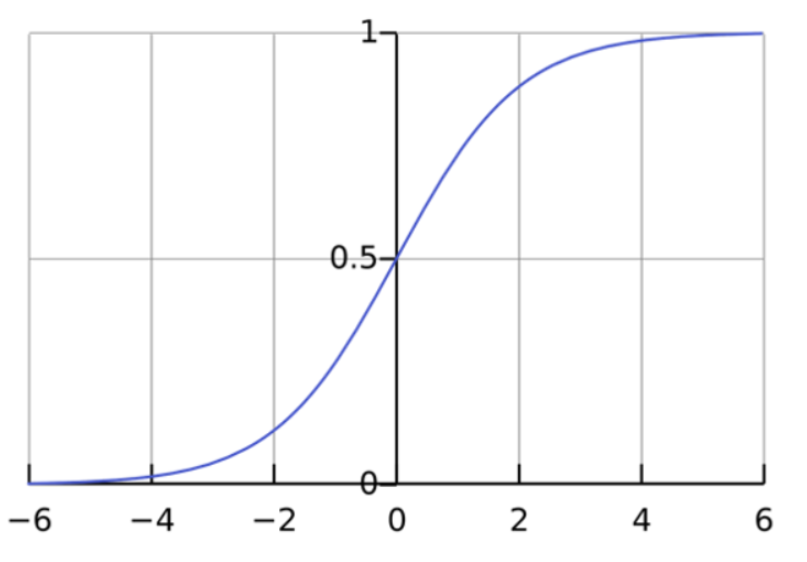
这就是logistic回归模型。

可以看出，这是一个sigmoid函数：



把h(x)作为自变量时的情形即为：

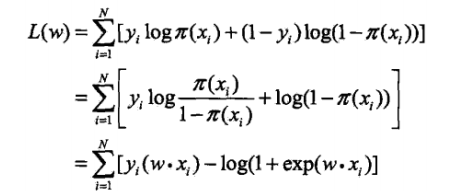


Sigmoid函数图象如图所示

**模型参数估计：**用极大似然估计法求模型参数

令

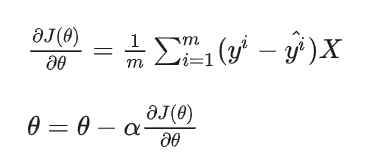
对数似然函数



**求L(w)的极大值点即为w估计值,我们采用梯度上升法求极大值。**

**这等同是交叉熵函数= -，采用梯度下降法求极小值**

（直接放公式图片了，这里为学习率）



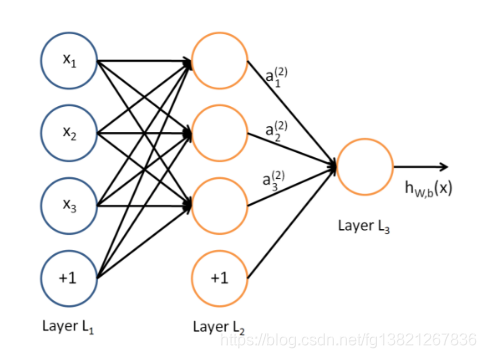
代码表示即

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](https://so.csdn.net/so/search?q=%E4%BA%BA%E5%B7%A5%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/fg13821267836/article/details/_blank)（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：

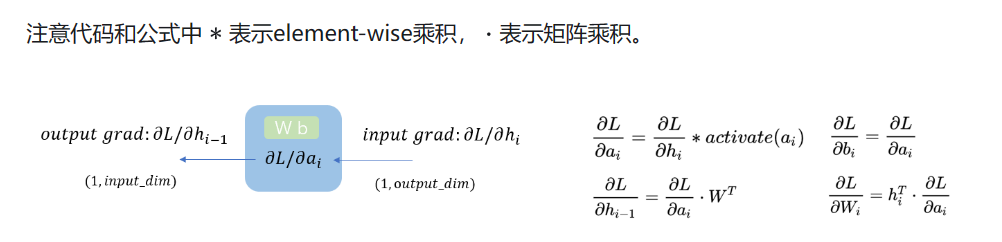
output\_hidden = sigmoid(W1X+b1)

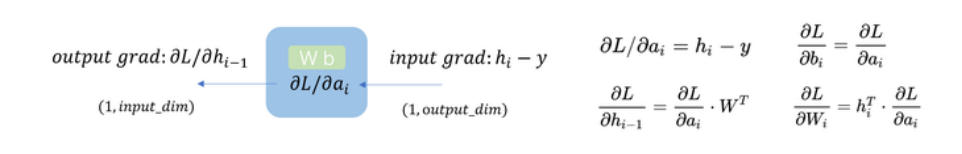
y\_hat = sigmoid(W2X+b2)

损失函数

Loss = mean(-sum(y\*log(y\_hat)-(1-y)\*log(1-y\_hat)))

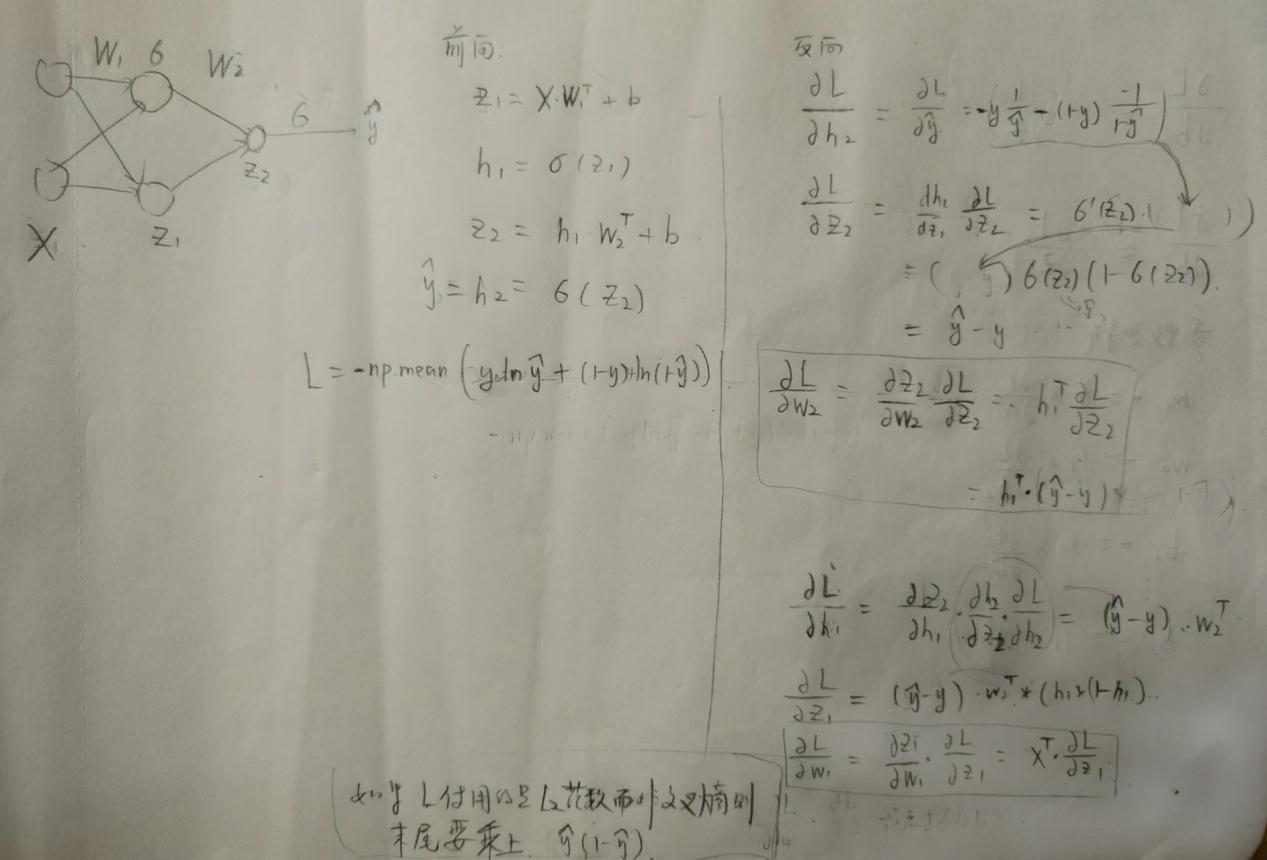
反向传播：

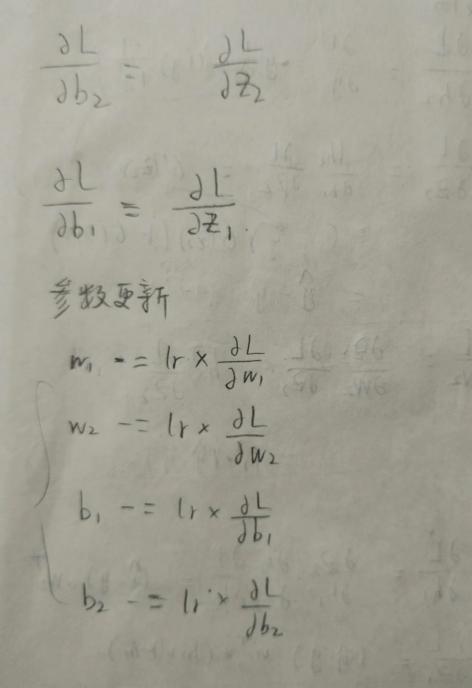




求梯度+更新权重：

手推了一下公式，不一定对





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。

1. 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)

1. 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

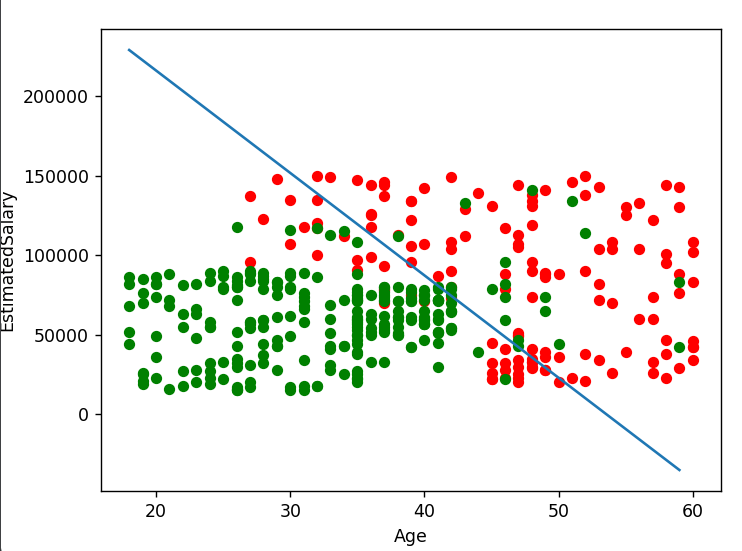
        out=mlp.forward(x\_data)

        loss = -np.mean(y\_data\*np.log(out)+(1-y\_data)\*np.log(1-out))

# 实验结果及分析

1. 实验结果展示示例

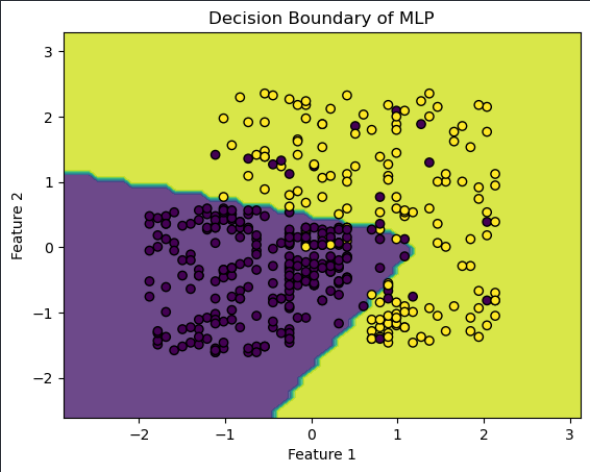
逻辑回归：





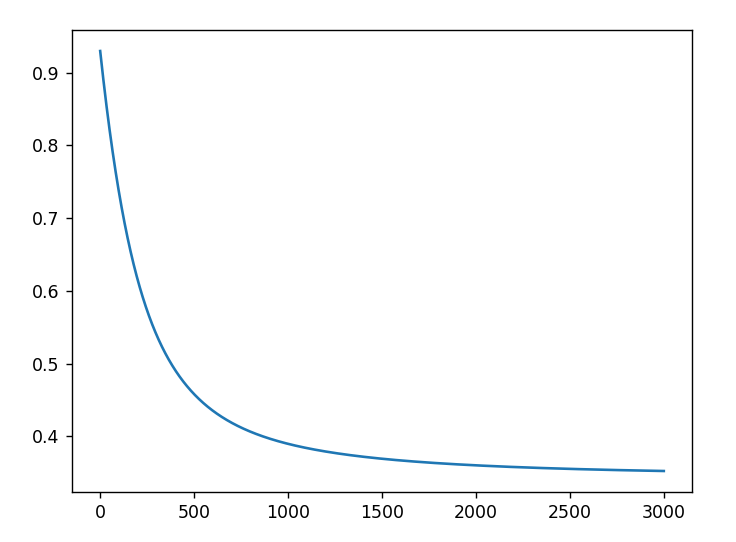
MLP(n\_hidden = 2):

横纵坐标值是归一化之后的特征的值

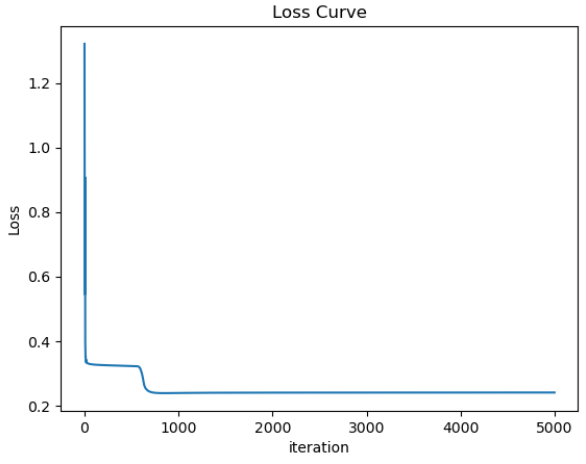


1. 评测指标展示及分析

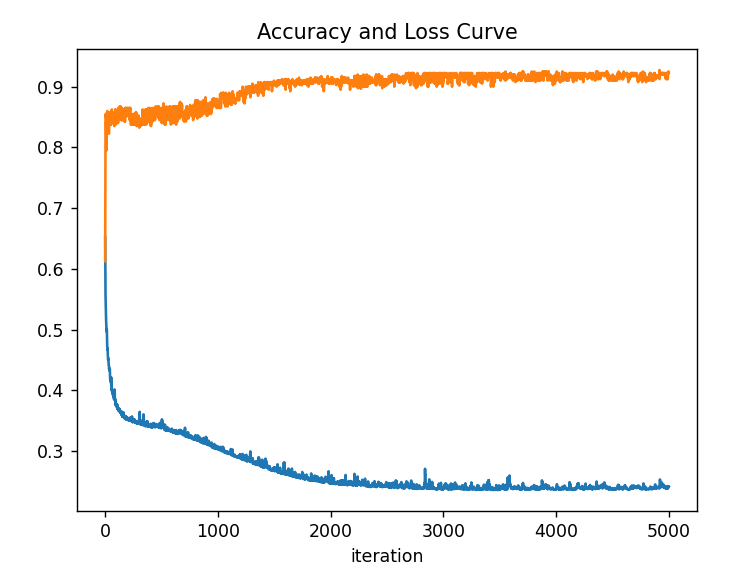
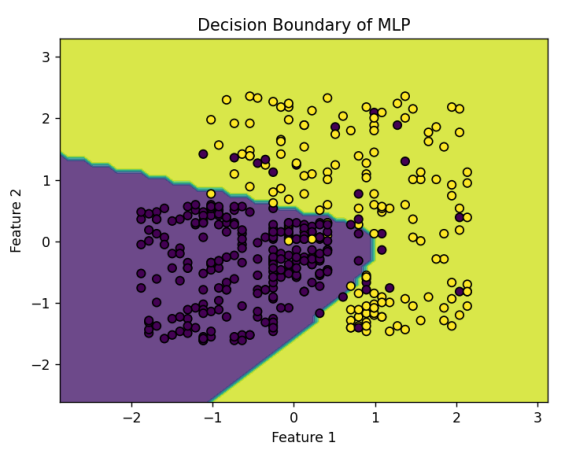
逻辑回归：



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。





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

# 参考资料

[LogisticRegression逻辑回归(附代码实现) - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/128102048)

[【Numpy】中np.random.rand()和np.random.randn()的用法和区别\_python np.random.randn-CSDN博客](https://blog.csdn.net/lemonxiaoxiao/article/details/109205647)

[sw\_machine\_learning/machine\_learning\_algorithm/logistic\_regression at master · yunshuipiao/sw\_machine\_learning (github.com)](https://github.com/yunshuipiao/sw_machine_learning/tree/master/machine_learning_algorithm/logistic_regression)

《统计学习方法》 李航

[深度学习 | 反向传播详解 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/115571464)

[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](https://blog.csdn.net/qq_43305312/article/details/115837064)