中山大学计算机学院人工智能本科生实验报告

课程：Artificial Intelligence
姓名： 学号：

一、实验题目

使用逻辑回归算法和感知机算法实现购房预测。

二、实验内容

1.算法原理

简便起见，只针对本问题所设置的情况说明算法原理，不针对一般情况。也就是，自变量只有两个，即年龄和收入，不妨分别记为x_1, x_2。因变量只有一个即是否购房，记为y，取值只有0或1。任务的数学化描述即是，寻找到合适的参数权重及偏置，使得$1, x_1, x_2$的线性组合经过非线性映射后能够最优地拟合实际取值。简便起见，将偏置b记为w_0，年龄和收入的权重分别为w_1, w_2。并记$z = w_0 + w_1x_1 + w_2x_2$。

(1)逻辑回归算法

逻辑回归中使用的非线性映射函数是$sigmoid$函数，其形式为：

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

此函数的取值范围为$(0,1)$，在一定程度上反映了给定样本取值为1的概率。而逻辑回归的数学原理就是概率论中的最大似然估计，即选择合适的参数使最大似然函数取得最大值。在将$sigmoid$函数看做概率的前提下，最大似然函数的形式是：

$$L(w) = \prod g(z_i)^{y_i} \cdot (1 - g(z_i))^{1-y_i}$$

> 因为为了求最大似然函数最大值必然涉及求导，非连续函数对求导操作非常不利，这就是选择$sigmoid$函数而不是符号函数作为非线性函数的原因。

最大化最大似然函数，等价于最小化损失函数。取对数不改变函数增减性，因此对$L(w)$取对数、取负、再按样本容量平均分，即得到交叉熵的损失函数：

$$l(w)=-\frac{1}{n}\sum y_i ln\ g(z_i)+(1-y_i)ln(1-g(z_i))$$

为了迭代得到合适的参数$w=[w_0,w_1,w_2]$，需要使各参数沿着其负梯度方向移动。为了求梯度——也就是求偏导数，不妨以$z_i=w_0+w_1x_{i1}+w_2x_{i2}$将损失函数展开为：

$$\begin{aligned}l(w_0,w_1,w_2)&=-\frac{1}{n}\sum -y_i ln(1+e^{z_i})+(1-y_i)(z_i-ln(1+e^{z}))\\&=-\frac{1}{n}\sum(1-y_i)z_i-ln(1+e^{z_i})\\&=\frac{1}{n}\sum ln(1+e^{w_0+w_1x_{i1}+w_2x_{i2}})-(1-y_i)(w_0+w_1x_{i1}+w_2x_{i2})\end{aligned}$$

于是：

$$\begin{aligned}\frac{\partial l(w)}{\partial w_0}&=\frac{1}{n}\sum\frac{z_i}{1+e_i^z}-(1-y_i)\\&=\frac{1}{n}\sum 1-g(z_i)+y_i-1\\&=\frac{1}{n}\sum y_i-g(z_i)\end{aligned}$$

$$\begin{aligned}\frac{\partial l(w)}{\partial w_1}&=\frac{1}{n}\sum\frac{x_{i1}z_i}{1+e_i^z}-(1-y_i)x_{i1}\\&=\frac{1}{n}\sum(y_i-g(z_i))x_{i1}\end{aligned}$$

$$\begin{aligned}\frac{\partial l(w)}{\partial w_2}&=\frac{1}{n}\sum\frac{x_{i2}z_i}{1+e_i^z}-(1-y_i)x_{i2}\\&=\frac{1}{n}\sum(y_i-g(z_i))x_{i2}\end{aligned}$$

然而梯度本身还带有大小，如果直接将梯度作为迭代的递减量，容易出现极值点附近震荡却不收敛的情况。因此使用“学习率”α来限制每一步递减的大小，即：

$$\begin{cases}w_0:=w_0-\alpha\cdot\frac{\partial l(w)}{\partial w_0}\\w_1:=w_1-\alpha\cdot\frac{\partial l(w)}{\partial w_1}\\w_2:=w_2-\alpha\cdot\frac{\partial l(w)}{\partial w_2}\end{cases}$$

这种迭代方式即是批量梯度下降，在求梯度时需要将所有样本都遍历一遍。有时为了节省时间，求梯度时可以选择所有样本中的一部分来遍历以近似求出梯度。

(2)感知机算法

与逻辑回归不同，感知机以符号函数作为激活函数：

$$s(z)=\begin{cases}1 & z\geq 0\\ -1 & z<0\end{cases}$$

由此，必然导致感知机与逻辑回归还有其他不同：

1. 由于符号函数的值域，损失函数不能使用交叉熵。感知机使用的损失函数是各(误分类)数据点到分类平面的距离之和。
2. 由于符号函数不连续，求导并不能很好地实现功能。也就是，虽然也是利用梯度下降来寻找最优的参数，但不可能使用逻辑回归中的批量梯度下降法。

对于感知机来说，重要的是被误分类的那些数据点。已经分类正确的可以忽略。这也使感知机使用批量梯度下降成为不可能。实际使用的是随机梯度下降，即随机选择其中一个误分类数据点，针对之调整参数以实现正确分类。

在这个问题中，分类直线是$w_0+w_1x_1+w_2x_2=0$。那么数据点与分类直线的距离显然就是：

$$d_i=\left|\frac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}\right|$$

对于正确分类的数据点来说，当$w_0+w_1x_{i1}+w_2x_{i2}>0$时恰有$y_i=1$，$w_0+w_1x_{i1}+w_2x_{i2}<0$时恰有$y_i=-1$，而误分类数据点则恰好相反。因此，为了在去掉绝对值符号的同时保证距离非负，只需改写为：

$$d_i=-y_i\times\frac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}$$

因此感知机的损失函数是：

$$loss(w)=\sum_{\text{误分类数据点}}-y_i\times\frac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}$$

有时为了提高效率，可以省略$\frac{1}{\sqrt{w_1^2+w_2^2}}$不计算，只计算：

$$loss(w)=\sum_{\text{误分类数据点}}-y_i\times(w_0+w_1x_{i1}+w_2x_{i2})$$

而对于随机梯度下降，只需要模仿逻辑回归算法中的梯度形式，将求和号去掉、只在误分类数据点中随机选择一个计算即可：

$$\begin{cases} \dfrac{\partial l(w)}{\partial w_0} & = y_i - s(z_i) \\ \dfrac{\partial l(w)}{\partial w_1} & = (y_i - s(z_i))x_{i1} \\ \dfrac{\partial l(w)}{\partial w_2} & = (y_i - s(z_i))x_{i2} \end{cases}$$

2.伪代码

(1)逻辑回归算法

首先给出算法大致框架的伪代码：

```
def Solve():
    for 迭代次数:
        计算损失函数值
        计算各自变量梯度
        各自变量 - 学习率 * 各梯度
    return 结果
```

然后给出类内结构：

```python
class LogicRegression:
    """逻辑回归算法"""
    def __init__(self, filename: str) -> None:

    def Probability(self, index: int) -> float:
        """计算给定样本取值为1的概率"""

    def Gradient(self) -> "tuple[float, float, float]":
        """求梯度"""

    def Loss(self) -> float:
        """使用交叉熵函数衡量损失"""

    def Accuracy(self) -> float:
        """计算预测准确率"""

    def Draw(self) -> None:
        """绘制结果"""

    def Solve(self) -> np.ndarray:
        """对外的解决方法"""

    @staticmethod
    def Sigmoid(input: float) -> float:

    @staticmethod
    def __Normalize__(array: np.ndarray) -> np.ndarray:
        """将给定的数组归一化"""
```

以下是函数调用层级：

$$Solve\begin{cases}Loss \to Probability \to Sigmoid \\ Gradient \to Probability \to Sigmoid \\ Draw \\ Accuracy \to Probability \to Sigmoid\end{cases}$$

函数实现是简单的。只需要将算法原理中的各式实现即可。

(2)感知机算法

由算法原理可以看出，感知机算法的整体框架与逻辑回归并无区别：

```
def Solve():
    for 迭代次数:
        计算损失函数值
        计算各自变量梯度
        各自变量 - 学习率 * 各梯度
    return 结果
```

然后给出类内结构，这与逻辑回归也是基本相似的：

```python
class Perceptron:
    def __init__(self, filename: str) -> None:

    def Loss(self) -> float:
        """使用距离衡量损失"""

    def Gradient(self) -> "tuple[float, float, float]":
        """求梯度"""

    def Accuracy(self) -> float:
        """计算预测准确率"""

    def Draw(self) -> None:
        """绘制"""

    def Solve(self) -> np.ndarray:
        """对外的解决方法"""

    @staticmethod
    def Sign(input: float) -> int:
        return (1) if (input >= 0) else (-1)

    @staticmethod
    def __Normalize__(array: np.ndarray) -> np.ndarray:
        """将给定的数组归一化"""
```

差别在于没有 `Probability` 方法，原因在于感知机的数学原理不在于最大似然估计。以下是函数调用层级：

$$Solve\begin{cases}Loss\\Gradient\rightarrow Sign\\Draw\\Accuracy\rightarrow Sign\end{cases}$$

函数实现是简单的。只需要将算法原理中的各式实现即可。

3.关键代码展示

(1)逻辑回归算法

```
def Solve(self) -> np.ndarray:
    """对外的解决方法"""
    for i in trange(0, self.iteration):
        self.loss_per_generation.append(self.Loss()) # 计算损失函数，同时记录各数据点的概率
        gradient = self.Gradient() # 计算梯度
        for index in range(0, len(self.weight)): self.weight[index] = self.weight[index] - sel
    self.Draw() # 绘制结果
    print("准确率：", self.Accuracy()) # 计算准确率
    return self.weight


def Loss(self) -> float:
    """使用交叉熵函数衡量损失"""
    loss = 0.0
    for index in range(0, self.capacity):
        self.probabilities[index] = self.Probability(index) # 更新记录数据点的概率
        loss = loss + self.purchased[index] * np.log(self.probabilities[index])
        loss = loss + (1 - self.purchased[index]) * np.log(1 - self.probabilities[index])
    return (-loss / self.capacity)


def Gradient(self) -> "tuple[float, float, float]":
    """求梯度"""
    grad_w0, grad_w1, grad_w2 = 0.0, 0.0, 0.0
    for index in range(0, self.capacity):
        temp = self.purchased[index] - self.probabilities[index] # 注意到梯度公式中有一个共同的项
        grad_w0 = grad_w0 + temp
        grad_w1 = grad_w1 + temp * self.__normalize_age__[index]
        grad_w2 = grad_w2 + temp * self.__normalize_estimateSalary__[index]
    return [grad_w0 / self.capacity, grad_w1 / self.capacity, grad_w2 / self.capacity]
```

(2)感知机算法

```
def Loss(self) -> float:
        """使用距离衡量损失"""
        loss, self.misclassified = 0.0, []
        for index in range(0, self.capacity):
            intermediate = self.weight[0] + self.weight[1] * self.__normalize_age__[index] + se
            if (intermediate > 0) and (self.purchased[index] == 0): # 如果是误分类点
                self.misclassified.append(index)
                loss = loss + intermediate
            elif (intermediate < 0) and (self.purchased[index] == 1): # 如果是误分类点
                self.misclassified.append(index)
                loss = loss - intermediate
        return loss / math.sqrt(self.weight[1] * self.weight[1] + self.weight[2] * self.weight[


    def Gradient(self) -> "tuple[float, float, float]":
        """求梯度"""
        chosen = self.misclassified[np.random.randint(0, len(self.misclassified))] # 随机选择一个
        intermediate = self.weight[0] + self.weight[1] * self.__normalize_age__[chosen] + self.
        grad_w0 = self.purchased[chosen] - Perceptron.Sign(intermediate) # 注意到另外两个梯度值都
        grad_w1, grad_w2 = grad_w0 * self.__normalize_age__[chosen], grad_w0 * self.__normalize_
        return [grad_w0, grad_w1, grad_w2]
```

4.创新优化

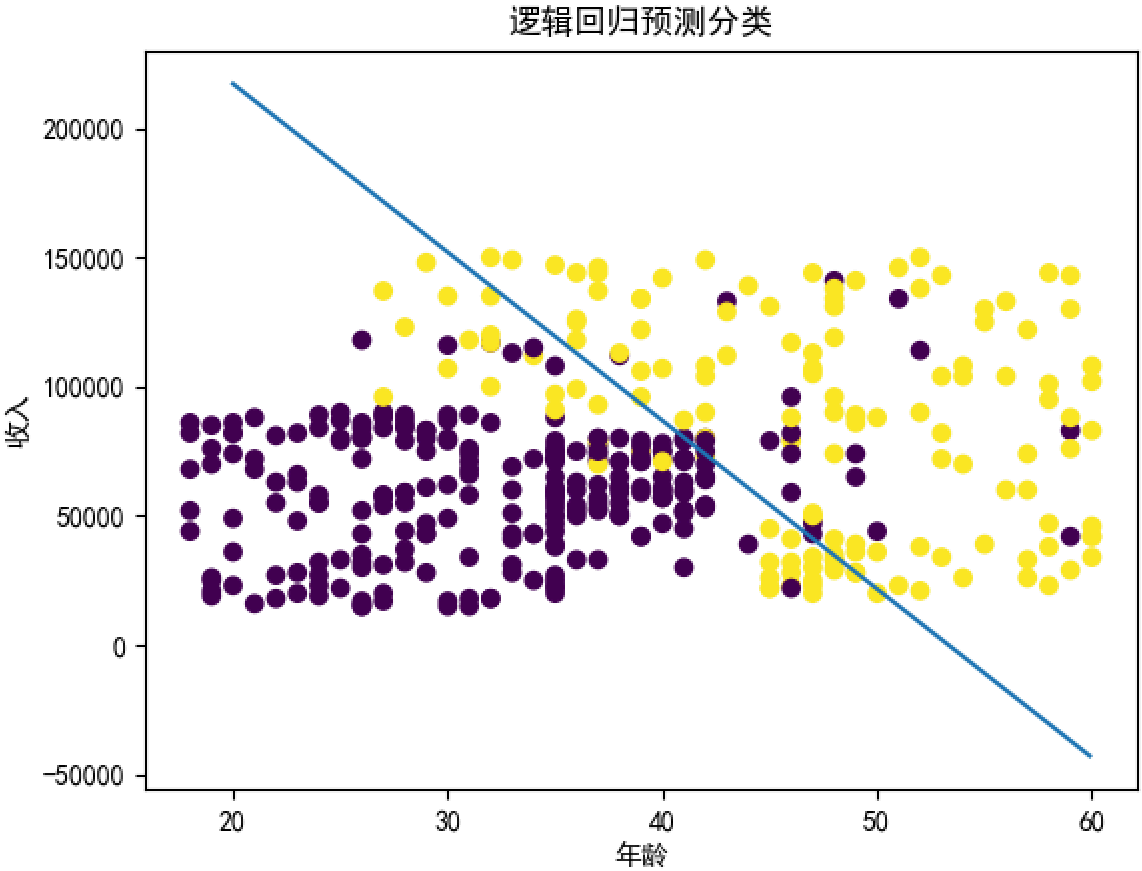
在逻辑回归算法中，可以发现 `Solve` 函数中要调用的函数在底层都是相同的，因此完全可以以空间换时间，在第一个调用的函数中就记录各数据点的“概率”，避免之后重复计算。

三、实验结果分析

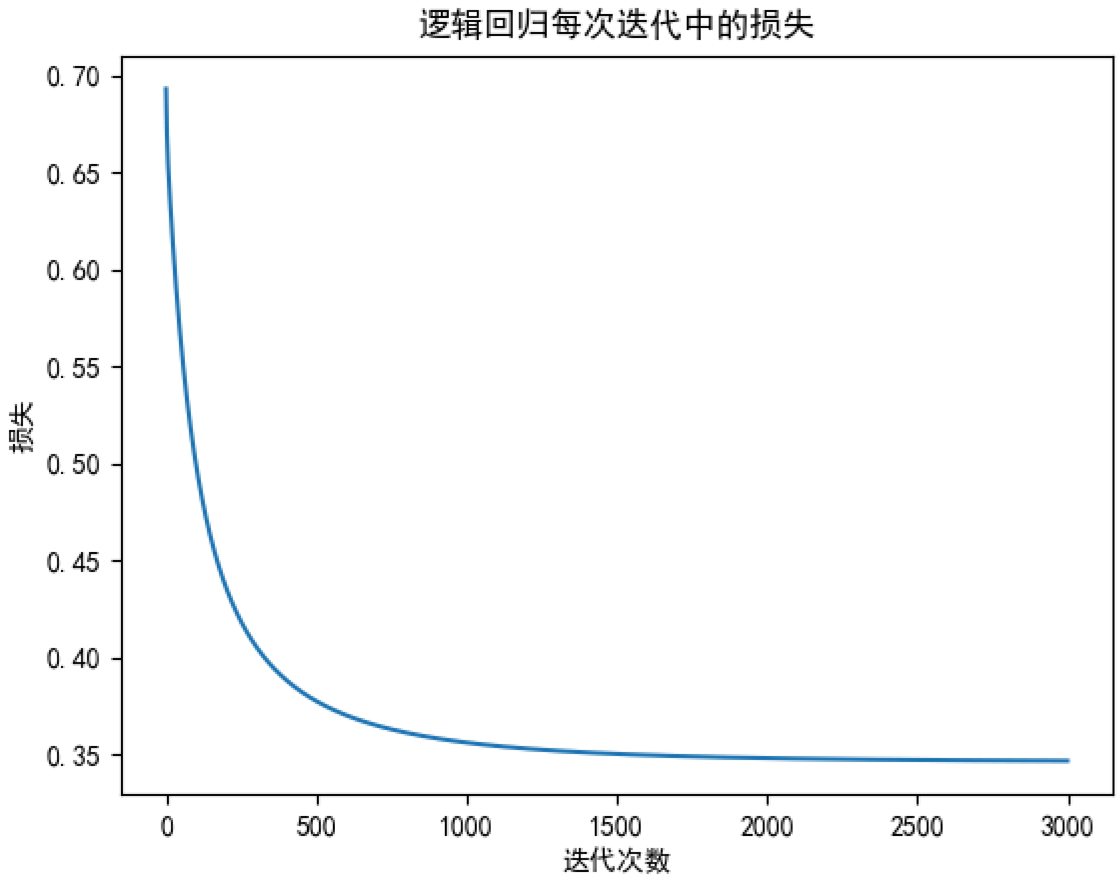
1.实验结果展示

(1)逻辑回归算法

首先是逻辑回归给出的分类线，以及数据集：

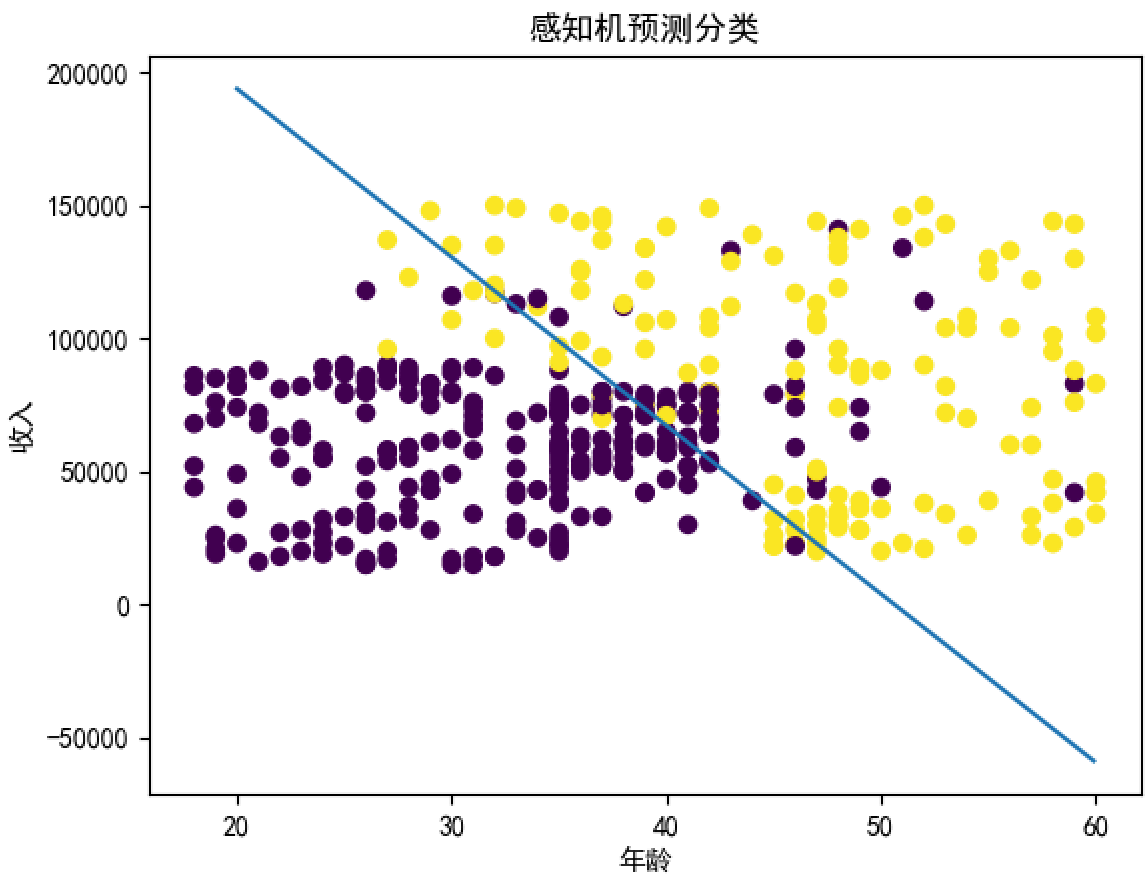


然后是损失函数随迭代次数的变化图：

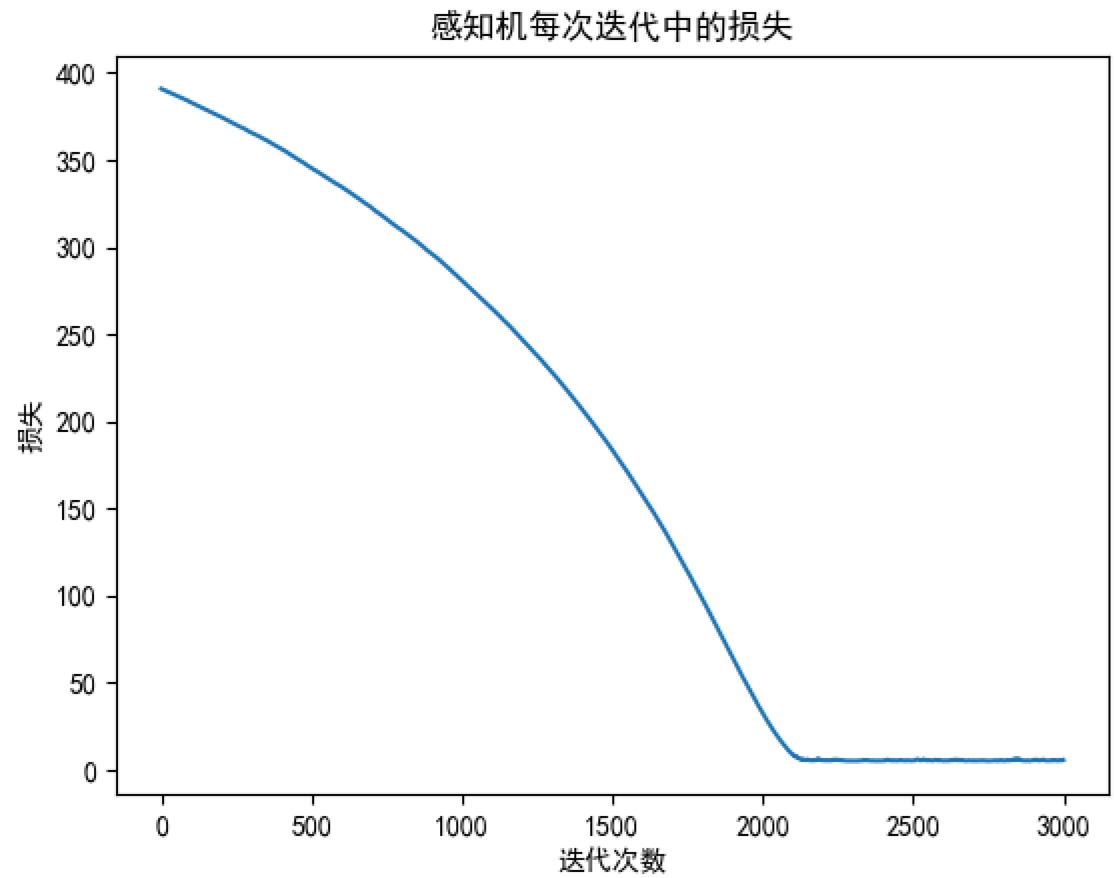


(2)感知机算法

首先是感知机给出的分类线，以及数据集：



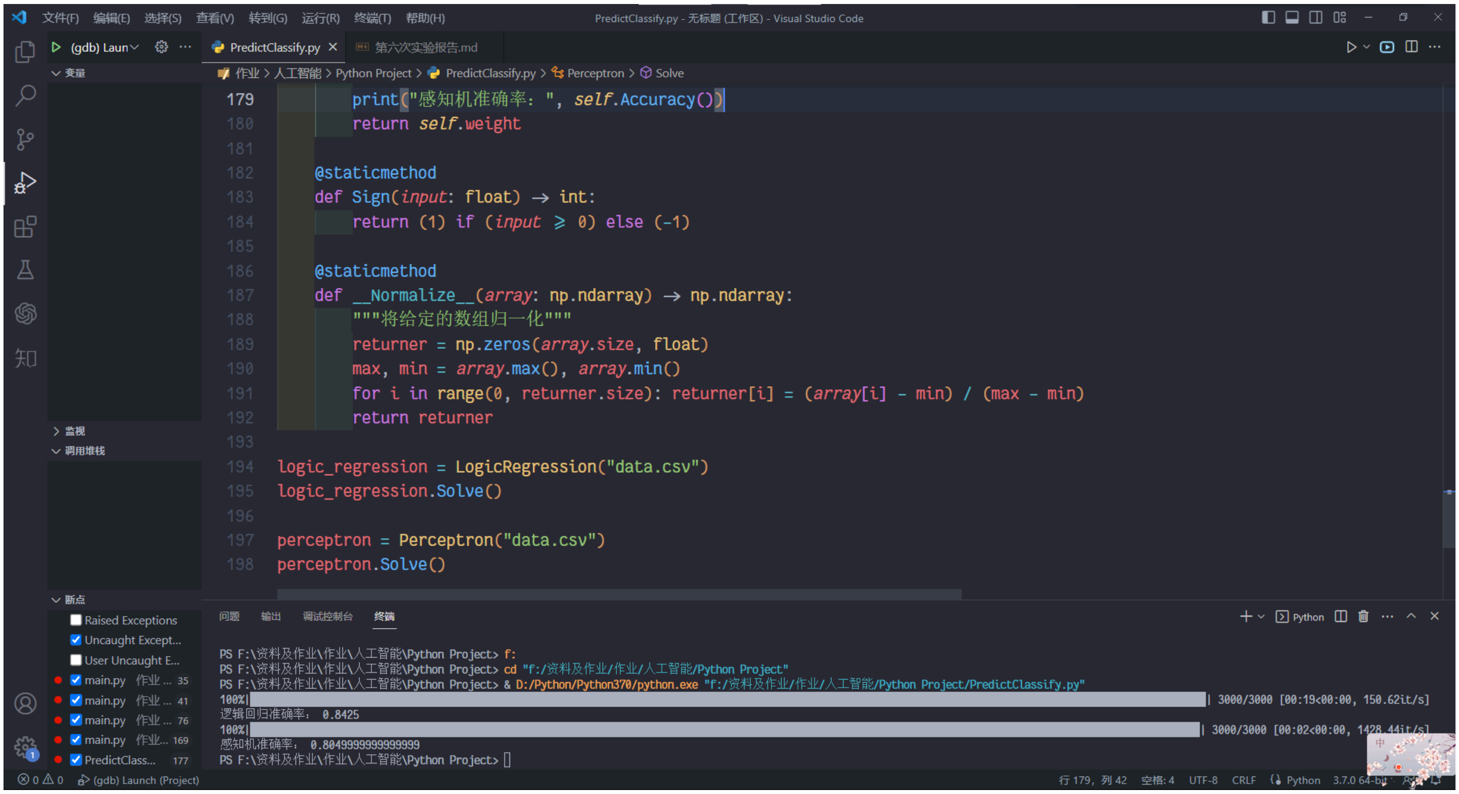
然后是损失函数随迭代次数的变化图：



两者的参数是：

	逻辑回归	**感知机**
迭代次数	3000	3000
学习率	0.5	0.0005

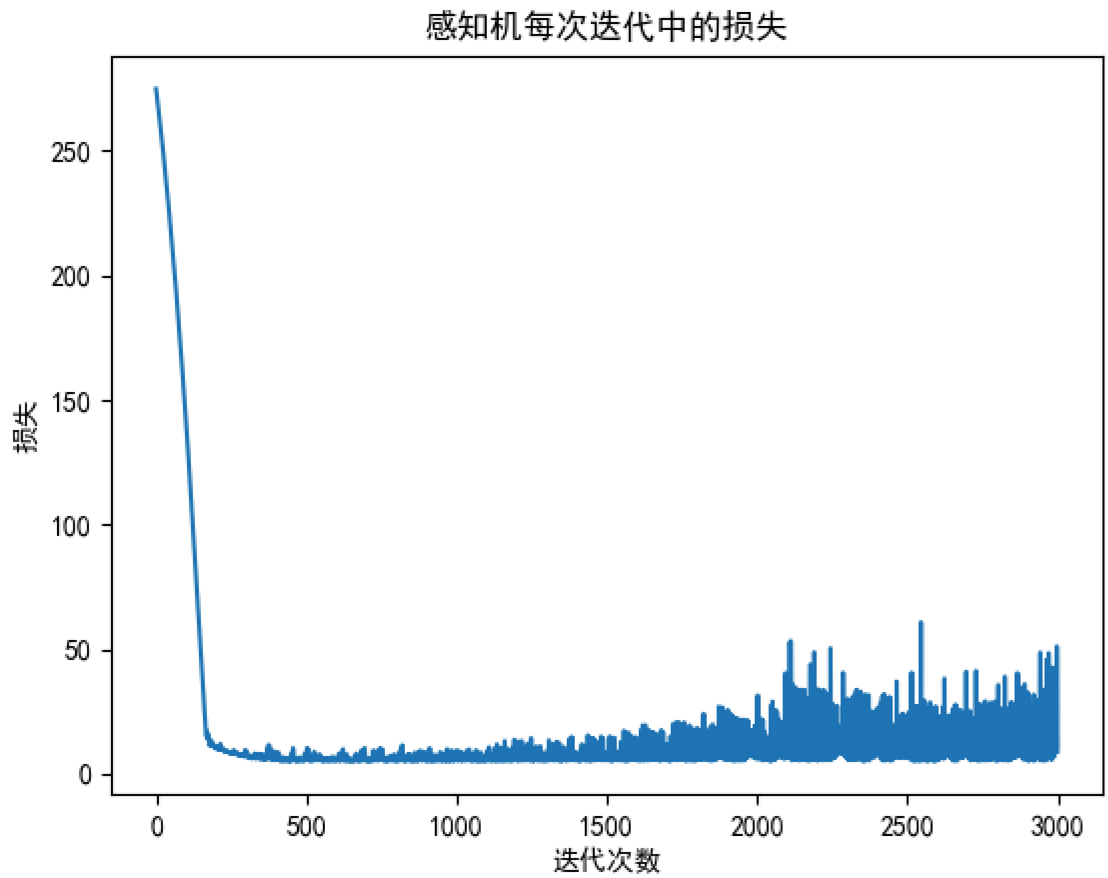
同时给出两者结束计算之后的准确率：



分别为84.25%和80.5%。

2.评测指标展示分析

观察可以发现，逻辑回归用时比感知机要多得多，分别为20秒左右和2秒左右。但感知机的准确率不如逻辑回归。实际上，感知机对学习率非常敏感，例如如果设置学习率为0.005时：



虽然损失函数一开始下降很快，但是到达极值点附近后，开始振荡，并且振幅有增加的趋势，使得最后一次迭代的结果不一定是最优结果。实际上，此次计算的准确率为77.25%。

四、思考题

无。

五、参考资料

Python实现逻辑回归(Logistic Regression)https://zhuanlan.zhihu.com/p/655259693