MLlab1 实验报告

PB19010450 和泳毅

一、 实验要求

本次实验要求完成逻辑回归的代码实现，并在给定数据集上进行训练和验证/测试。具体需要完成以下部分：

- 读取训练数据集和测试数据集对数据进行预处理
- 初始化逻辑回归模型
- 实现优化算法
- 在训练数据集上进行模型的参数优化，要求该步骤在有限时间内停止
- 在测试数据集上进行测试，并输出测试集中样本的预测结果

二、 数据集介绍

Data Set Characteristics:	Multivariate	Number of Instances:	569	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	32	Date Donated	1995-11-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	1607794

该数据集是从乳腺肿块的细针抽吸（FNA）的数字化图像计算特征。 它们描述了图像中存在的细胞核的特征。

共569个样本，每个样本有30个特征。其中M为恶性占37%，B 为良性占63%。

ID	Label	f1	f2	f3	f4	f5	f6	f7	f8	...	f21	f22	f23	f24	f25	f26	f27	f28	f29	f30
842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

三、 实验原理

1. 模型

对于线性模型：

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b \quad \text{s.t.} f(\boldsymbol{x}) \approx y,$$

其中$\boldsymbol{w}$为权重，b为偏置，$\boldsymbol{x}$为属性，$\boldsymbol{y}$为标签。

推广到广义线性模型有:

$$f(\boldsymbol{x}) = g^{-1}(\boldsymbol{w}^T\boldsymbol{x}+b) \quad \text{s.t.} f(\boldsymbol{x}) \approx y,$$

其中g是链接函数，单调可微。

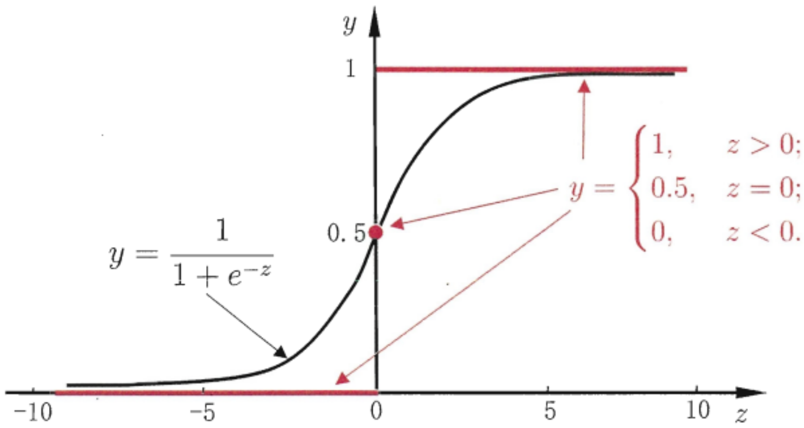
本实验关注线性模型在分类问题上的一个典型应用：**对数几率回归**(Logistic Regression) 。

有基本形式:

$$f(\boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}} \quad \text{s.t.} f(\boldsymbol{x}) \approx y,$$

其中$y \in \{0,1\}$，即逻辑回归是一个二分类模型。其广义线性模型是一个特例:

$$g(y) = \ln\frac{y}{1-y} = \boldsymbol{w}^T\boldsymbol{x}+b.$$



2. 优化目标

在分类任务中，采用最大似然法，最大化模型的对数似然函数:

$$\ell(\boldsymbol{w},b) = \sum_{i=1}^{m} y_i \log P(y=1|\boldsymbol{x}_i;\boldsymbol{w},b) + (1-y_i)\log P(y=0|\boldsymbol{x}_i;\boldsymbol{w},b)$$

$$P(y=1|\boldsymbol{x};\boldsymbol{w},b) = \hat{y} = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

$$P(y=0|\boldsymbol{x};\boldsymbol{w},b) = 1 - P(y=1|\boldsymbol{x};\boldsymbol{w},b)$$

由于$\ell(\boldsymbol{w},b)$是一个关于$(\boldsymbol{w},b)$的高阶可导连续凸函数，可用经典的数值优化算法求全局最优解。

3.优化方法

负对数似然:

$$\ell(\hat{\boldsymbol{w}}) = \sum_{i=1}^{m}(-y_i\hat{\boldsymbol{w}}^T\hat{\boldsymbol{x}}_i) + \log(1+e^{\hat{\boldsymbol{w}}^T\hat{\boldsymbol{x}}_i})).$$

一阶导数:

$$\nabla \ell(\hat{\boldsymbol{w}}) = -\sum_i (y_i - P(y=1|\hat{\boldsymbol{x}}_i;\hat{\boldsymbol{w}}))\hat{\boldsymbol{x}}_i = -\sum_i (y_i - p_1)\hat{\boldsymbol{x}}_i$$

二阶导数:

$$\nabla^2 \ell(\hat{\boldsymbol{w}}) = \frac{\partial \nabla \ell(\hat{\boldsymbol{w}})}{\partial \hat{\boldsymbol{w}}^T} = \sum_i p_1(1-p_1)\hat{\boldsymbol{x}}_i \hat{\boldsymbol{x}}_i^T$$

可以采用梯度下降法或牛顿法实现目标函数的优化:

- **梯度下降法**

$$\boxed{\begin{array}{l} \text{while } \|\nabla \ell(\hat{\boldsymbol{w}})\| > \epsilon \quad do \\ \quad \hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k - \alpha \nabla \ell(\hat{\boldsymbol{w}}_k) \\ \text{end while} \end{array}}$$

- **牛顿法**

$$\boxed{\begin{array}{l} \text{while } \|\nabla \ell(\hat{\boldsymbol{w}})\| > \epsilon \quad do \\ \quad \hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k - (\nabla^2 \ell(\hat{\boldsymbol{w}}_k))^{-1} \nabla \ell(\hat{\boldsymbol{w}}_k) \\ \text{end while} \end{array}}$$

改进牛顿法——DFP法:

牛顿法虽然有二阶收敛速率，但是使用牛顿法需要不断地计算二阶导，而且目标函数的Hesse矩阵$G_k = \nabla^2 \ell(\hat{\boldsymbol{w}}_k)$可能不正定，甚至非奇异。于是引入拟牛顿法，用不含二阶导数的矩阵H_k近似牛顿法中的Hesse矩阵的逆G_k^{-1}。

设第$k+1$次迭代后得到$\hat{\boldsymbol{w}}_{k+1}$，将目标函数$\ell(\hat{\boldsymbol{w}})$在$\hat{\boldsymbol{w}}_{k+1}$处二阶Taylor展开，有:

$$\ell(\hat{\boldsymbol{w}}) \approx \ell(\hat{\boldsymbol{w}}_{k+1}) + \nabla \ell(\hat{\boldsymbol{w}}_{k+1})^T(\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}_{k+1}) + \frac{1}{2}(\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}_{k+1})^T \nabla^2 \ell(\hat{\boldsymbol{w}}_{k+1})(\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}_{k+1}),$$

进一步有:

$$\nabla \ell(\hat{\boldsymbol{w}}) \approx \nabla \ell(\hat{\boldsymbol{w}}_{k+1}) + \nabla^2 \ell(\hat{\boldsymbol{w}}_{k+1})(\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}_{k+1}),$$

于是令$\hat{\boldsymbol{w}} = \hat{\boldsymbol{w}}_k$，有:

$$\nabla \ell(\hat{\boldsymbol{w}}_k) \approx \nabla \ell(\hat{\boldsymbol{w}}_{k+1}) + \nabla^2 \ell(\hat{\boldsymbol{w}}_{k+1})(\hat{\boldsymbol{w}}_k - \hat{\boldsymbol{w}}_{k+1}).$$

记$\boldsymbol{s}_k = \hat{\boldsymbol{w}}_{k+1} - \hat{\boldsymbol{w}}_k$，$\boldsymbol{y}_k = \nabla \ell(\hat{\boldsymbol{w}}_{k+1}) - \nabla \ell(\hat{\boldsymbol{w}}_k)$，则有:

$$\nabla^2 \ell(\hat{\boldsymbol{w}}_{k+1})\boldsymbol{s}_k \approx \boldsymbol{y}_k \quad \text{or} \quad \nabla^2 \ell(\hat{\boldsymbol{w}}_{k+1})^{-1}\boldsymbol{y}_k \approx \boldsymbol{s}_k$$

这样计算出$\boldsymbol{s}_k$和$\boldsymbol{y}_k$后，可依上式估计在$\hat{\boldsymbol{w}}_{k+1}$处的Hesse矩阵的逆，在迭代中构造出Hesse矩阵逆的近似H_{k+1}，使其满足:

$$H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k.$$

称作正割条件，也称作拟牛顿条件。

设H_k为第k次迭代的Hesse矩阵逆的近似，我们希望以H_k来产生H_{k+1}，即

$$H_{k+1}=H_k+E_k,$$

其中E_k是一个低秩的矩阵。为此采用对称秩二（SR2）校正：

$$H_{k+1}=H_k+a\boldsymbol{u}\boldsymbol{u}^T+b\boldsymbol{v}\boldsymbol{v}^T,$$

并使得正割条件成立，有:

$$H_{k+1}\boldsymbol{y}_k=H_k\boldsymbol{y}_k+(a\boldsymbol{u}^T\boldsymbol{y}_k)\boldsymbol{u}+(b\boldsymbol{v}^T\boldsymbol{y}_k)\boldsymbol{v}=\boldsymbol{s}_k$$

取特解:

$$\begin{cases}\boldsymbol{u}=\boldsymbol{s}_k, & a\boldsymbol{u}^T\boldsymbol{y}_k=1;\\ \boldsymbol{v}=H_k\boldsymbol{y}_k, & b\boldsymbol{v}^T\boldsymbol{y}_k=-1.\end{cases}$$

因此有:

$$H_{k+1}=H_k+\frac{\boldsymbol{s}_k\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}-\frac{H_k\boldsymbol{y}_k\boldsymbol{y}_k^TH_k}{\boldsymbol{y}_k^TH_k\boldsymbol{y}_k}.$$

上式称为DFP(Davidon-Fletcher-Powell)校正公式。

算法可以描述如下:

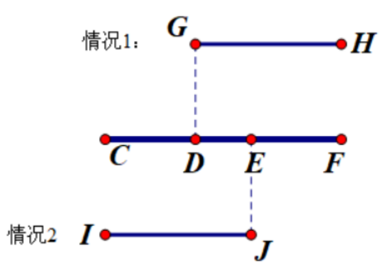
$$\boxed{\begin{array}{l}H_0=I\\ \text{while}\ \|\nabla\ell(\hat{\boldsymbol{w}})\|>\epsilon\quad do\\ \quad \boldsymbol{d}_k=-H_k\nabla\ell(\hat{\boldsymbol{w}}_k)\\ \quad \hat{\boldsymbol{w}}_{k+1}=\hat{\boldsymbol{w}}_k+\alpha\boldsymbol{d}_k\\ \quad H_{k+1}=H_k+\dfrac{\boldsymbol{s}_k\boldsymbol{s}_k^T}{\boldsymbol{s}_k^T\boldsymbol{y}_k}-\dfrac{H_k\boldsymbol{y}_k\boldsymbol{y}_k^TH_k}{\boldsymbol{y}_k^TH_k\boldsymbol{y}_k}\\ \text{end while}\end{array}}$$

确定学习率的方法:

无论梯度下降法还是DFP拟牛顿法，更新权值都需要学习率，如果固定学习率，则需要在训练时不断地调整。如果学习率过大可能导致无法收敛，过小可能导致收敛过慢。考虑到该实验的样本数不大，可以使用一维搜索确定学习率。

- 精确一维搜索

  黄金分割法:



设黄金分割比为k，在区间(a,b)内取$\alpha_1=b-k(b-a)$，$\alpha_2=a+k(b-a)$。在一维搜索中，若$f(\alpha_1)<f(\alpha_2)$，说明极小值点在α_1的右侧，修改搜索区间为(α_1,b)。设$\rho=1-k$，则$CD=1-\rho, DE=\rho, EF=1-\rho$。在黄金分割比的作用下：

$$\frac{1-\rho}{1}=\frac{1-2\rho}{1-\rho},$$

因此

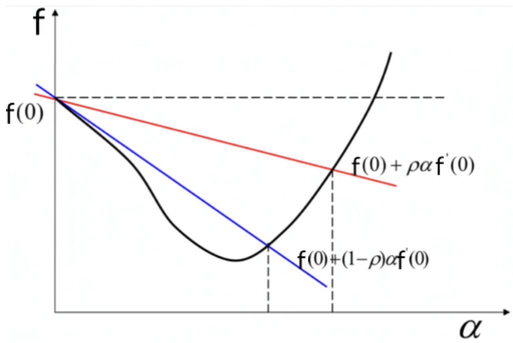
$$\frac{DE}{GH}=\frac{DE}{DF}=\frac{DF}{CF}=k\approx 0.618$$

因此若将E点平移至GH线段，将G的横坐标当做a，H的横坐标当做b，它就相当于是α_1的位置，只需要计算$\alpha_2=a+k(b-a)=b-\rho(b-a)$即可。若$f(\alpha_1)>f(\alpha_2)$同理讨论。

- 非精确一维搜索

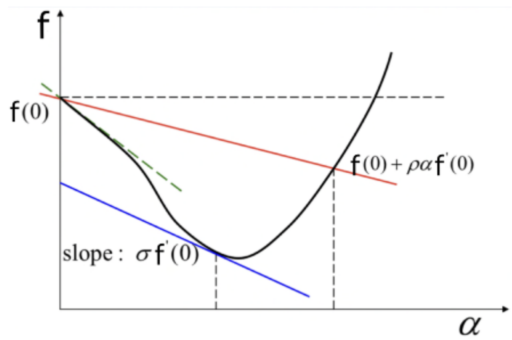
  Goldstein条件：

  $$f(\alpha)\le f(0)+\rho\alpha f'(0),\quad f(\alpha)\ge f(0)+(1-\rho)\alpha f'(0)\quad \rho\in(0,1/2)$$

  Wolfe-Powell条件：

  $$f(\alpha)\le f(0)+\rho\alpha f'(0),\quad f'(\alpha)\ge\sigma f'(0)\quad \rho\in(0,1/2),\sigma\in(\rho,1)$$

由于本实验样本数不大，可以考虑采用一维精确搜索来确定最佳单步学习率。

四、核心代码讲解

Sigmoid函数：

```
def sigmoid(x):
    if x>=0:
        return 1.0/(1+np.exp(-x))
    else:
        return np.exp(x)/(1+np.exp(x))
```

模型参数：

```
 '''参数描述
    lr：梯度更新的学习率
    Lambda：收敛条件
    epochs：更新迭代的次数
    w：训练参数
    status_dict：标签列表
    maxmin：归一化数据
    '''
```

- **lr**：模型的学习率，即优化目标函数的步长，默认值为0.05。
- **Lambda**：判断收敛条件的阈值，默认值为0.01。
- **epochs**：迭代的最大次数，默认值为1000。
- **w**：训练得到的参数，包括权重和偏置。
- **status_dict**：存放用于标签转换的列表，初始化为['B','M']。
- **maxmin**：存放训练归一化的参数，包括每个特征的最大值最小值。

数据预处理：

```
def pretreatment(self, df, flag='train'):
        df_c = df.copy()
        #添加一列
        df_c['f31']=1
        if flag=='train':
            #转换标签
            df_c['Label_tran']=df_c['Label'].apply(lambda x : self.status_dict.index(x))
        #特征归一化
        for i in range(30):
            if flag == 'train':
                self.maxmin += [[df_c.iloc[:,i+2].min(axis=0),df_c.iloc[:,i+2].max(axis=0)]]
            df_c.iloc[:,i+2]=(df_c.iloc[:,i+2]-self.maxmin[i][0])/(self.maxmin[i][1]-self.maxmin[i][0])
        return df_c
```

为了将偏置加入权重列表一起训练，需要在特征矩阵最后添加一列1。同时，在该问题中，我们关注的是将恶性样本分类出来，所以将M转化为正例1，将B转化为反例0。最后对特征做归一化：$\frac{x_{ij}-\min \boldsymbol{x}_i}{\max \boldsymbol{x}_i-\min \boldsymbol{x}_i}$，将每个特征的最大值最小值存为模型参数，预测时用该参数对测试集特征做归一化处理，保证每个特征有同样的大小关系。

黄金分割法一维搜索：

```python
def gold_div_search(a,b,esp,N,w,x,y,dleta):
    # 黄金分割法一维搜索，返回学习率值
            rou = 1 -(sqrt(5) - 1) / 2 # 1-rou为黄金分割比
            lr1 = a + rou * (b - a)
            lr2 = b - rou * (b - a)
            while(b-a > esp):
                w1 = w - lr1 * dleta
                w2 = w - lr2 * dleta
                w1_T_x = np.dot(w1.T[0], x)
                w2_T_x = np.dot(w2.T[0], x)
                f1 = 0
                f2 = 0
                for i in range(N):
                    f1 = f1 + (-y[i] * w1_T_x[i] + np.log(1 +
np.exp(w1_T_x[i])))
                    f2 = f2 + (-y[i] * w2_T_x[i] + np.log(1 +
np.exp(w2_T_x[i])))
                if f1 > f2:  #如果f1>f2，则在区间(lr1,b)内搜索
                    a = lr1
                    lr1 = lr2
                    lr2 = b - rou * (b - a)
                elif f1 < f2:  #如果f1<f2,则在区间(a,lr2)内搜索
                    b = lr2
                    lr2 = lr1
                    lr1 = a + rou * (b - a)
                else:  #如果f1=f2，则在区间(lr1,lr2)内搜索
                    a = lr1
                    b = lr2
                    lr1 = a + rou * (b - a)
                    lr2 = b - rou * (b - a)
            return a
```

训练部分：

- 使用梯度下降：

```python
def fit(self, train_features, train_labels):

```

```
        def gold_div_search(a,b,esp,N,w,x,y,dleta):
            '''...'''

        x = np.array(train_features).T
        y = np.array(train_labels)
        num, N = np.shape(x)
        w = np.ones((num, 1))
        L=np.zeros(self.epochs+1)
        L[0]=1000

        for j in range(self.epochs):
            w_T_x = np.dot(w.T[0], x)
            # 似然函数
            for i in range(N):
                L[j+1] = L[j+1] + (-y[i] * w_T_x[i] + np.log(1 + np.exp(w_T_x[i])))
            # 收敛条件
            if np.abs(L[j+1] - L[j]) <= self.Lambda:
                break
            # 计算梯度
            dbeta = 0
            for i in range(N):
                p1 = sigmoid(w_T_x[i])
                dbeta = dbeta - np.array([x[:, i]]).T * (y[i] - p1)
            gk = dbeta
            # 优化参数
            self.lr = gold_div_search(0,1,0.005,N=N,w=w,x=x,y=y,dleta=gk)
            w = w - self.lr * gk
            epochs = j+1
        self.w = w  # 存储训练参数
```

收敛条件选择为两次目标函数值变化不超过self.Lambda，否则按照最大迭代次数self.epochs退出。通过一维搜索得到单步最佳学习率进行优化。

- 使用拟牛顿法DFP

```
    def fit(self, train_features, train_labels):

        def gold_div_search(a,b,esp,N,w,x,dleta):
            '''...'''

        x = np.array(train_features).T
        y = np.array(train_labels)
        num, N = np.shape(x)
```

```python
        w = np.ones((num, 1))
        L=np.zeros(self.epochs+1)
        L[0]=1000
        Hk = np.eye(31)

        for j in range(self.epochs):
            w_T_x = np.dot(w.T[0], x)
            # 收敛条件
            for i in range(N):
                L[j+1] = L[j+1] + (-y[i] * w_T_x[i] + np.log(1 + np.exp(w_T_x[i])))
            if np.abs(L[j+1] - L[j]) <= self.Lambda:
                break
            # 计算梯度与下降方向
            dbeta = 0  # 一阶导
            for i in range(N):
                p1 = sigmoid(w_T_x[i])
                dbeta = dbeta - np.array([x[:, i]]).T * (y[i] - p1)
            gk = dbeta
            dk = -1.0*np.dot(Hk,gk)
            # 优化参数
            self.lr = gold_div_search(0,1,0.005,N=N,w=w,x=x,y=y,dleta=-dk)
            w_new = w + self.lr * dk
            # 更新Hk
            sk = w_new - w
            w_new_T_x = np.dot(w_new.T[0], x)
            dbeta = 0
            for i in range(N):
                p1 = sigmoid(w_new_T_x[i])
                dbeta = dbeta - np.array([x[:, i]]).T * (y[i] - p1)
            yk = dbeta - gk
            if np.dot(sk.T,yk) > 0:
                Hy = np.dot(np.dot(Hk,yk),yk.T)
                sy = np.dot(sk.T,yk)
                yHy = np.dot(np.dot(yk.T,Hk),yk)
                Hk = Hk - np.dot(Hy,Hk)/yHy + np.dot(sk,sk.T)/sy
            w = w_new
            epochs = j+1
        self.w = w  # 存储训练参数
```

H_0为31维的单位阵，每次迭代需要计算梯度g_k和下降方向d_k。通过一维搜索得到单步最佳学习率进行优化。最后通过DFP公式更新H_k来近似Hesse矩阵的逆。

预测部分：

```
 def predict(self, test_features):
        x=np.array(test_features).T
        w_T_x=np.dot(self.w.T[0],x)
        pre=np.zeros((x.shape[1],1))
        y=np.zeros((x.shape[1],1))

        for i in range(pre.shape[0]):
            pre[i] = sigmoid(w_T_x[i])
            if pre[i] < 0.5:
                y[i] = 0
            elif pre[i] > 0.5:
                y[i] = 1
            else:
                y[i] = random.randint(0,1)

        y_label=pd.DataFrame(y)
        y_label= y_label.iloc[:,0].apply(lambda x : self.status_dict[int(x)])
        y_label=np.array(y_label)
        return y_label
```

通过训练参数计算测试集样本的几率，并转换为相应的正负例标签。

评价指标：

```
# 计算TP,FP,FN
def get_binary_TP_FP_FN(y,pred,label):
    num=y.shape[0]
    # pred=pred.astype(int).tolist()
    TP=0
    FP=0
    FN=0
    TN=0
    for i in range(num):
        if y[i][0]==label and pred[i][0]==label:
            TP+=1
        elif y[i][0]==label and pred[i][0]!=label:
            FN+=1
        elif y[i][0]!=label and pred[i][0]==label:
            FP+=1
        else:
            TN+=1
```

```python
    return TP,FP,FN

# 根据TP,FP,FN计算得到P,R
def get_binary_P_R(TP,FP,FN):
    return TP/(TP+FP),TP/(TP+FN)

# 根据P,R计算得到对应的F1-score
def get_binary_f1(P,R):
    return 2*P*R/(P+R)

# 计算准确率
def get_acc(y,pred):
    return np.sum(y==pred)/len(y)

# 计算二分类的F1
def get_F1(y,pred,label):
    TP,FP,FN = get_binary_TP_FP_FN(y,pred,label)
    P,R = get_binary_P_R(TP,FP,FN)
    F1 = get_binary_f1(P,R)
    return P,R,F1
```

五、实验中遇到的问题及解决方案

- 矩阵运算维度匹配问题

  矩阵的维度不匹配时会使得算法无法运行，所以要通过左乘矩阵、右乘矩阵或者加入转置运算来使得维度匹配。在遇到矩阵维度不匹配问题时，可以将所有用到的矩阵的维度记录在纸上，手动检查矩阵乘法。

- 算法收敛性问题

  刚开始采用向量的1范数或者2范数作为收敛判断条件，但发现阈值调参不直观，并且导致收敛速度过慢。于是采用两次损失函数的差作为收敛条件。

- 预处理归一化问题

  如果对测试集特征采用测试集的特征最大值最小值进行归一化，可能会导致测试集数据对训练得出的w有不严谨的大小关系的表达。例如：训练集中，某特征最大值为100；测试集中该特征最大值为150。两个最大值都被归一化为1，w对他们的敏感度一样，但150要大于100。于是保存训练集特征的归一化参数，用于测试集特征归一化。

- 固定学习率使得损失函数随着训练产生波动

  对一个模型固定一个学习率，可能会导致不收敛、收敛速度慢等情况。并且使得损失函数下降曲线产生较大波动，也大大增加调参的工作量。由于该实验数据量不大，可以采用精确一维搜索——黄金分割法。

- 牛顿法Hesse阵迭代几次后不正定甚至非奇异

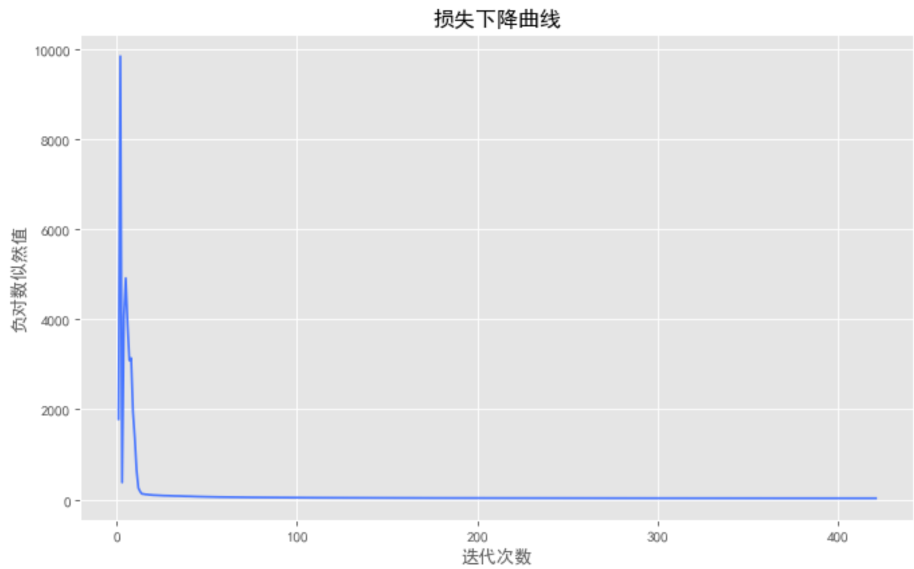
  尝试使用牛顿法下降，发现迭代几次后Hesse矩阵非奇异，下降无法进行。于是采用拟牛顿法DFP公式，用H_k来近似Hesse矩阵的逆，避免了直接的运算。

六、实验结果

1.对比固定学习率与一维搜索确定学习率

采用给定训练集数据与梯度下降学习法。

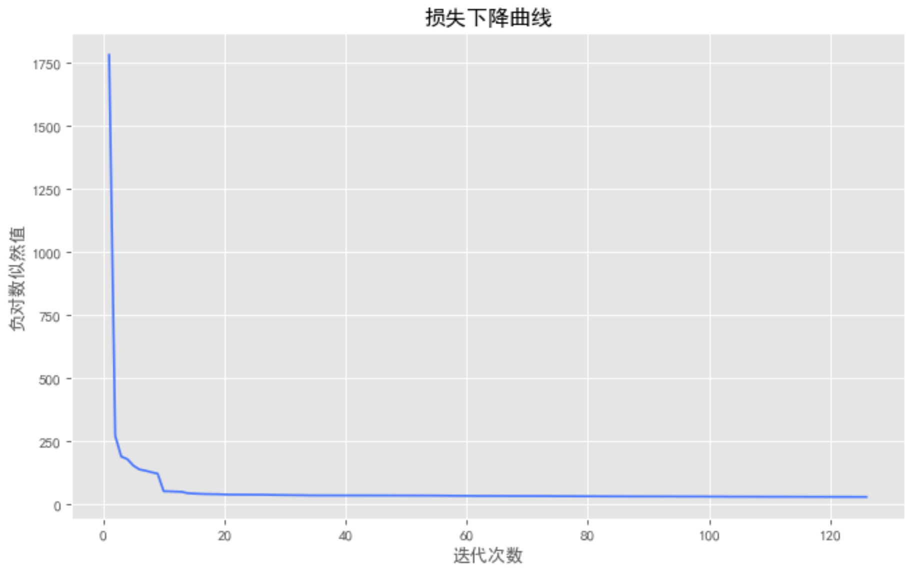
若采用固定学习率，以准确率为标准进行调参，选取lr=0.08。




```
训练 421 次，在训练集上:
准确率: 0.9824175824175824
查准率: 0.9828571428571429
查全率: 0.9717514124293786
F1指标: 0.9772727272727272
```

可以看到损失函数存在产生较大波动的情况。

若采用一维搜索确定学习率：



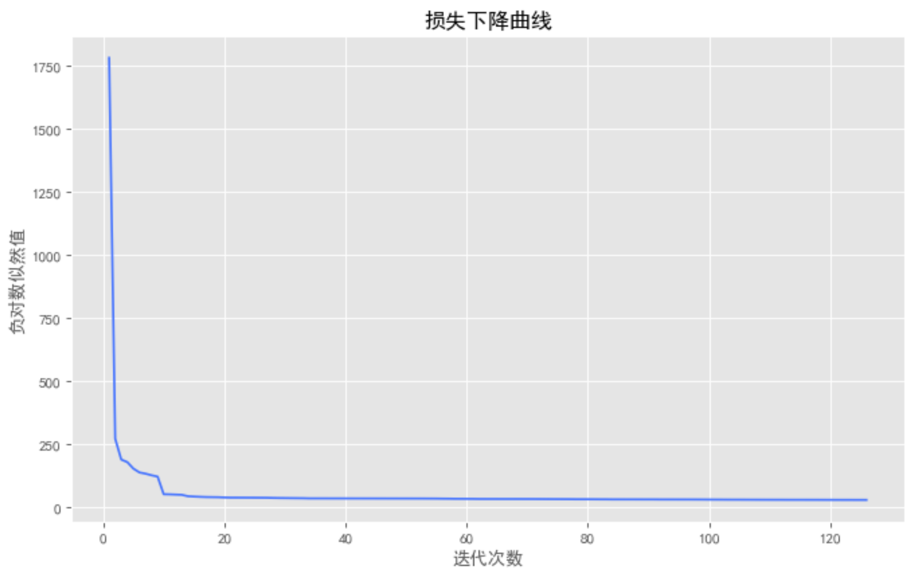

```
训练 126 次，在训练集上：
准确率： 0.9846153846153847
查准率： 0.9885057471264368
查全率： 0.9717514124293786
F1指标： 0.98005698005698
```

可以看到损失函数下降曲线变得平滑，并且各项评价指标都有提升，收敛次数减小。

2.对比梯度下降法与拟牛顿法

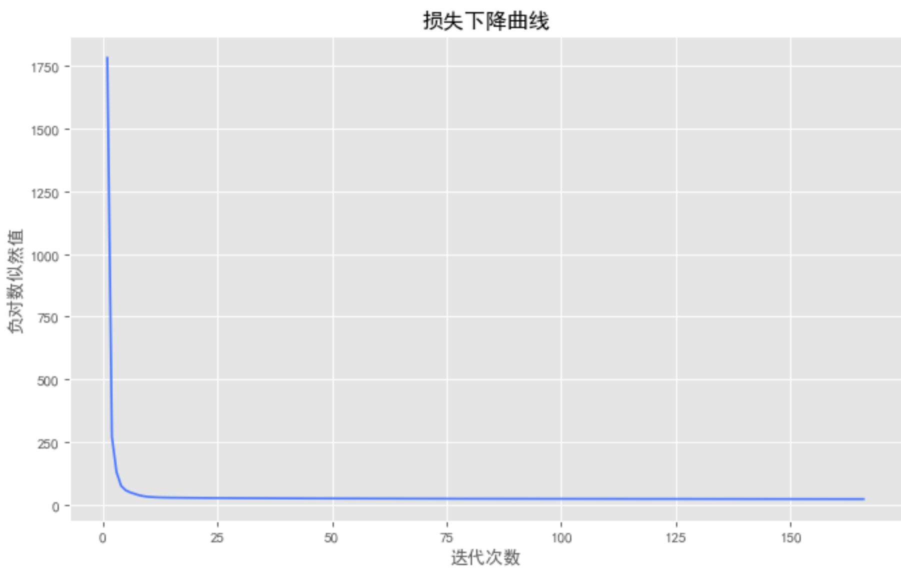
采用给定训练集数据与一维搜索确定学习率。

若采用梯度下降法:




```
训练 126 次，在训练集上：
准确率： 0.9846153846153847
查准率： 0.9885057471264368
查全率： 0.9717514124293786
F1指标： 0.98005698005698
```

若采用拟牛顿法DFP:




```
训练 166 次，在训练集上:
准确率： 0.9802197802197802
查准率： 0.9772727272727273
查全率： 0.9717514124293786
F1指标： 0.9745042492917847
```

采用拟牛顿法后的各项评价指标都有一定的下降，并且由于数据量不大，收敛速度的优势没有体现出来。

3.训练与测试

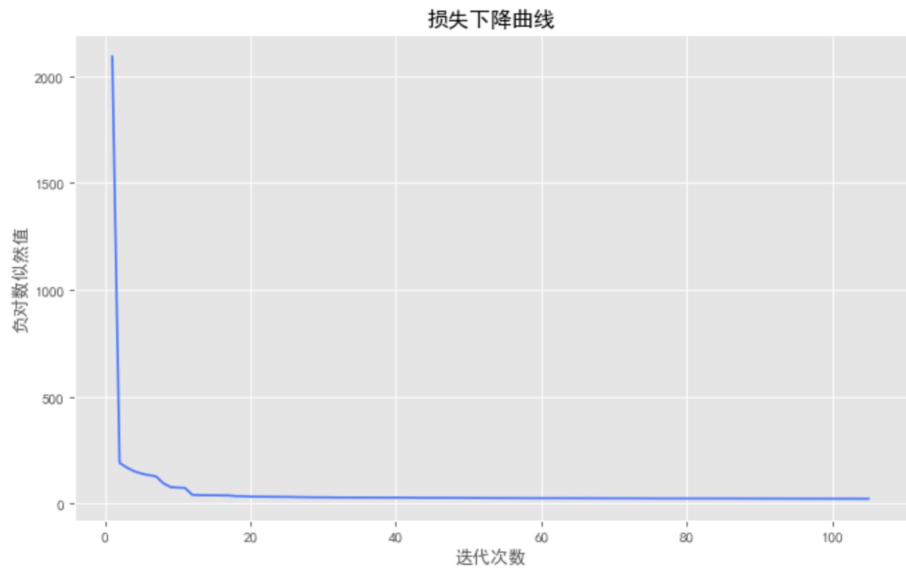
根据前面的对比试验，正式测试采用一维搜索确定学习率的梯度下降法。

采用5折交叉验证，划分数据集：

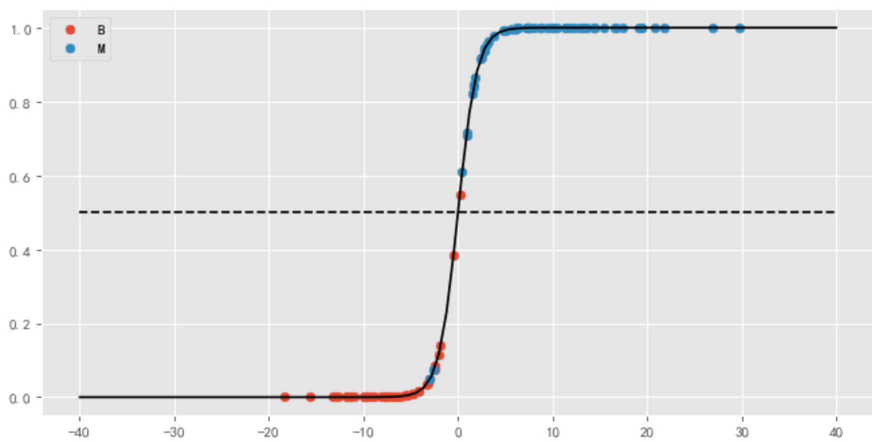
```
names=['ID','Label']
for i in range(30):
    names.append('f'+str(i+1))
df =  pd.read_csv("wdbc.data",names=names)

for i in range(5):
    df2=df[i*114:i*114+114]
    df1=pd.concat([df[0:i*114],df[i*114+114:569]])
    '''...'''
```

第一折：

损失下降曲线图：



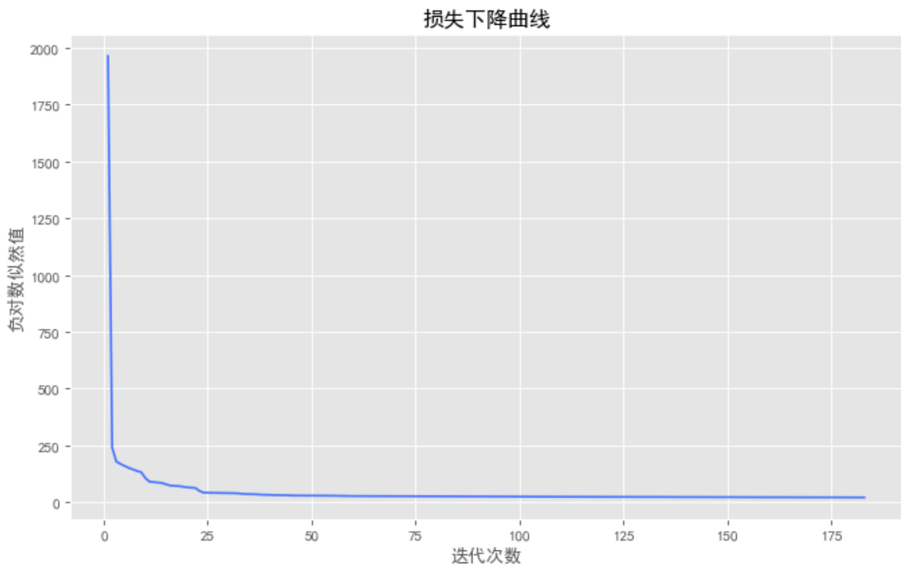
测试集分类结果:



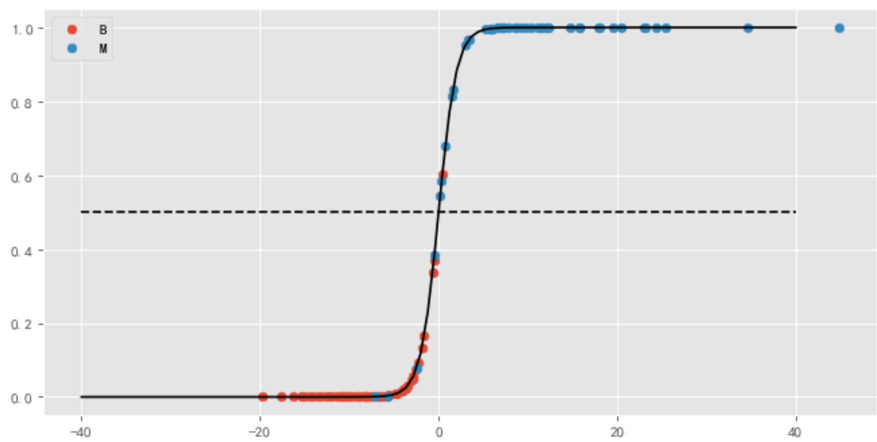
训练 105 次，在训练集上：
准确率： 0.989010989010989
查准率： 1.0
查全率： 0.9652777777777778
F1指标： 0.9823321554770319
在测试集上：
准确率： 0.9649122807017544
查准率： 0.9848484848484849
查全率： 0.9558823529411765
F1指标： 0.9701492537313432

第二折:

损失下降曲线图:



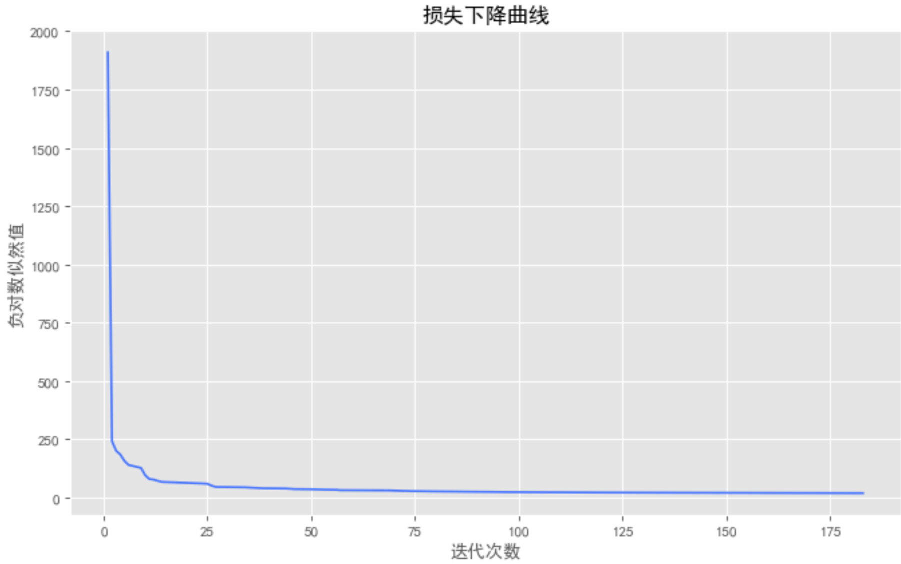
测试集分类结果：



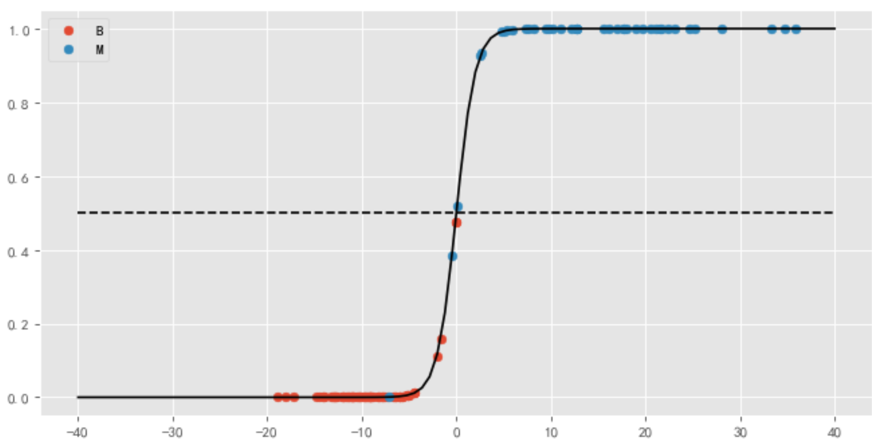
训练 183 次，在训练集上：
准确率： 0.9912087912087912
查准率： 1.0
查全率： 0.9754601226993865
F1指标： 0.9875776397515528
在测试集上：
准确率： 0.956140350877193
查准率： 0.9782608695652174
查全率： 0.9183673469387755
F1指标： 0.9473684210526316

第三折：

损失下降曲线图：



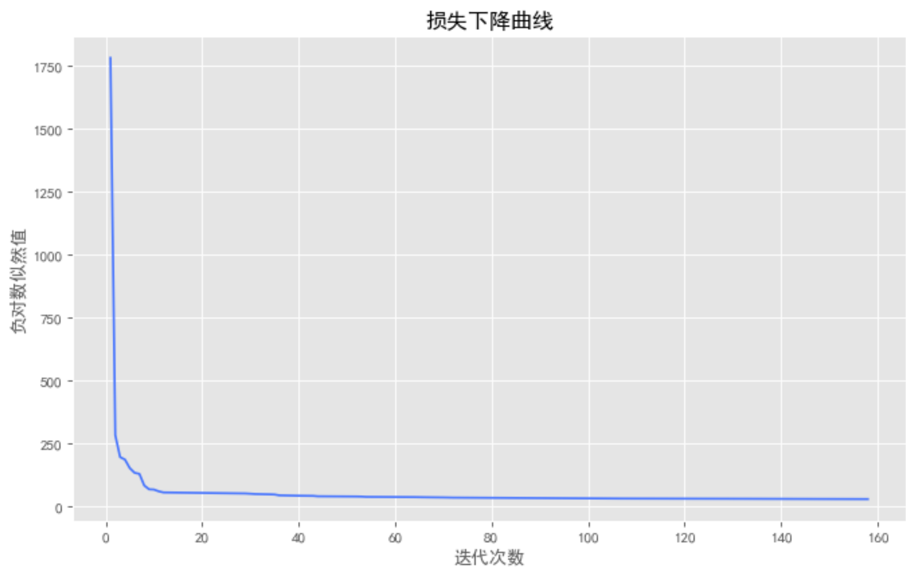
测试集分类结果:



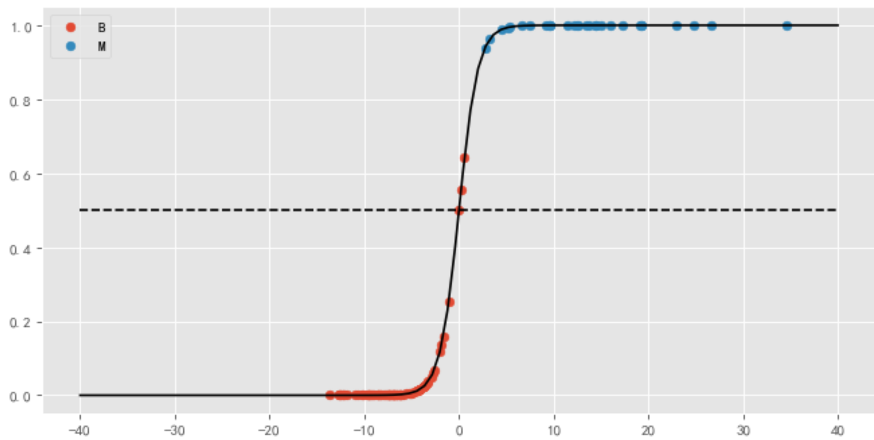
训练 183 次，在训练集上:
准确率： 0.989010989010989
查准率： 0.9883040935672515
查全率： 0.9825581395348837
F1指标： 0.9854227405247813
在测试集上:
准确率： 0.9824561403508771
查准率： 1.0
查全率： 0.95
F1指标： 0.9743589743589743

第四折:

损失下降曲线图:



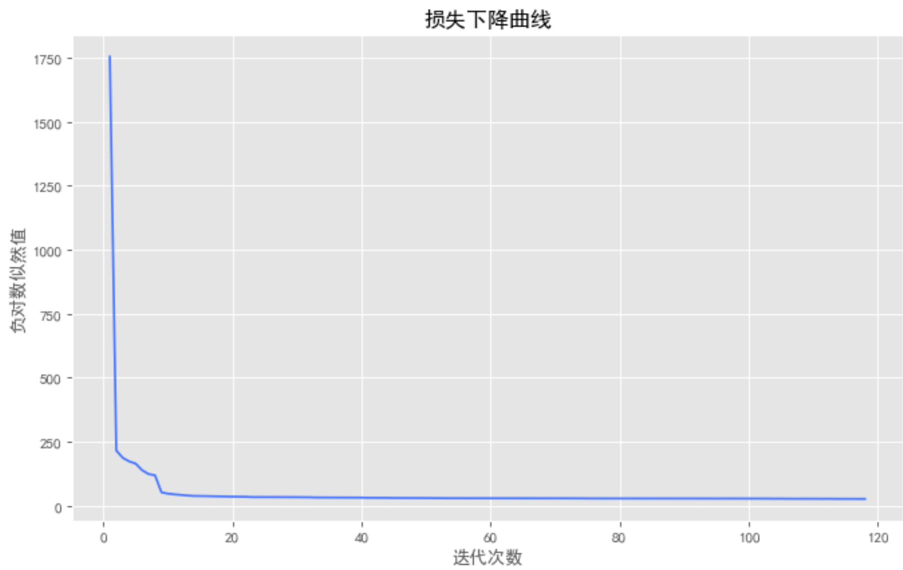
测试集分类结果:



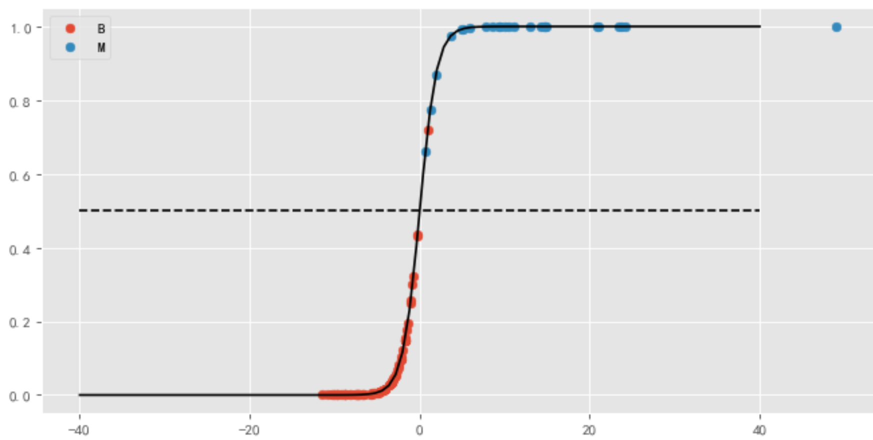
训练 158 次，在训练集上:
准确率: 0.9868131868131869
查准率: 0.988950276243094
查全率: 0.9781420765027322
F1指标: 0.9835164835164836
在测试集上:
准确率: 0.9824561403508771
查准率: 0.9354838709677419
查全率: 1.0
F1指标: 0.9666666666666666

第五折:

损失下降曲线图:



测试集分类结果：



训练 118 次，在训练集上：
准确率： 0.9802631578947368
查准率： 0.9783783783783784
查全率： 0.9731182795698925
F1指标： 0.9757412398921833
在测试集上：
准确率： 0.9911504424778761
查准率： 0.9629629629629629
查全率： 1.0
F1指标： 0.9811320754716981

则最后求得在测试集上的评价指标：

平均准确率： 0.9754230709517155
查准率： 0.9723112376688814
查全率： 0.9648499399759904
F1指标： 0.9679350782562628

在本问题中，我们需要预测Breast Cancer的恶性或良性，应用时我们更应该关注恶性的样本是否能被预测出来。所以我们在预处理时将恶性样本标记为正例，评价此模型不仅要关注其准确率，还要关注其查全率。

七、结论

该逻辑回归模型采用一维搜索确定学习率进行梯度下降优化，经5折交叉验证，平均准确率有97.54%，查全率有96.48%。