ML 实验报告

——logistic 回归

PB19020499　　桂栋南

目录

1. 实验要求

1.1. 实验数据集

1. Breast cancer Wisconsin (diagnostic) dataset （威斯康星州乳腺癌（诊断）数据集）。
2. 其中数据数量为 569 条数据，30 个属性（无缺失值），二分类标签。
3. 数据存放在.data 文件中，数据以文本形式给出，可直接通过文本形式读入。
4. 数据以‘，’分隔，均为字符串格式，处理数据时，需要将数据转换为 float 格式。
5. 数据的标签为第二列，‘M’、‘B’为两种标签，处理数据时需要将其转换为 0-1 数据。

1.2. 实验具体要求

1. 通过命令行格式运行.py 文件，将数据文件软编码，通过 sys.argv 获取文件路径。
2. 输出预测结果，即对于 test.data 的数据的每行进行预测，并将预测结果（‘M’or‘B’）打印。
3. 在文件中对于 train.data 数据进行训练，对 test.data 的数据进行预测，要求预测正确率>=90%。
4. 预测结果的准确率(acc)作为评价指标。

2. 实验原理

二分类问题，采用 logistic 函数解决。即求参数$\boldsymbol{\beta}$有

$$f(x)=\frac{1}{1+e^{-(w^\top x+b)}}=\frac{1}{1+e^{\beta^\top x}}\ \text{s.t.}\ f(x)\approx y$$

问题的优化目标是极大似然函数。即

$$\mathrm{l}(w,b)=\sum_{i=1}^{m} y_i\log P\,(y=1|x_i;w,b)+(1-y_i)\log P\,(y=0|x_i;w,b)$$

为了实现的方便，转为极小化极大似然函数的负对数，即负对数似然函数。即

$$\mathrm{l}(\widehat{w})=\sum_{i=1}^{m}\left(-\,y_i\widehat{w^\top\widehat{x_i}}+\log\left(1+e^{\widehat{w^\top\widehat{x_i}}}\right)\right)$$

优化方法采用梯度下降的方法实现。即

$$\widehat{w_{t+1}}\leftarrow\widehat{w_t}-\alpha\nabla\mathrm{l}(\widehat{w})$$

其中有

$$\nabla\mathrm{l}(\widehat{w})=\sum_i f^{'}(z_i)\frac{\partial z_i}{\partial\widehat{w}}=\sum_i\left(-\,y_i+\frac{1}{1+e^{-z_i}}\right)x_i=-\sum_i\left(y_i-P(y=1|\widehat{x_i};\widehat{w})\right)\widehat{x_i}$$

$$=-\sum_i(y_i-p_1)\widehat{x_i}$$

3. 实验实现（代码讲解）

3.1. 文件读入与处理

1. 读取数据：

```
train_path = str(sys.argv[1])
test_path = str(sys.argv[2])

with open(train_path,'r') as fp:
    train_data = fp.readlines()

with open(test_path,'r') as fp:
    test_data = fp.readlines()
```

2. 获取训练集与测试集，读入 dataframe，确定训练数据和标签以及测试数据：

```
16 item_list = []
17 for item in train_data:
18     item = item[:-1]
19     item = item.split(',')
20     for i in range(len(item[2:])):
21         try:
22             item[i+2] = float(item[i+2])
23         except:
24             pass
25     item_list.append(item)
26 df = pd.DataFrame(item_list)
27 
28 df.replace(['M','B'],[0,1],inplace=True)
29 X = df.loc[:,2:]
30 y = df.loc[:,1]
```

20~24 行：将数据转为 float 格式；28 行将标签转化为 0-1 标签。
测试集同理：

```
item_list = []
for item in test_data:
    item = item[:-1]
    item = item.split(',')
    for i in range(len(item[2:])):
        try:
            item[i+2] = float(item[i+2])
        except:
            pass
    item_list.append(item)
df_test = pd.DataFrame(item_list)

X_test = df_test.loc[:,2:]
# y_test = df_test.loc[:,1]
```

3.2. Logistic-regression 实现

3.2.1. 大体结构

```
48  class LogisticRegression():
49      def __init__(self,dim):...
52
53      def sigmoid(self,x):...
55
56      def normalization(self,X):...
66
67      def gradient_decrease(self,X,y,alpha=0.01,epoch=1000,epsilon=1e-3):...
92
93      def fit(self,X):...
```

定义了 LogisticRegression 类来处理问题。

1. Init 类初始化权重，需要数据的维度。
2. Sigmoid 就是一个 sigmoid 函数，方便后续调用。
3. normalization 正则化函数，对输入的数据矩阵的每一列进行正态归一化。
4. gradient_decrease 梯度下降，需要设置学习率 alpha，迭代次数 epoch，停止间隔 epsilon。
5. fit 对测试数据进行预测。

3.2.2. 归一化

```
def normalization(self,X):
    #有些数据过大，进行正则化，进行正态分布的正则化
    m,n = X.shape
    for k in range(n):
        i = k + 2
        temp = X.loc[:,i]
        avg = np.average(temp)
        var = np.var(temp)
        X.loc[:,i] = (X.loc[:,i] - avg)/ var**0.5
    return X
```

有些数据过于大，送入 sigmoid 后会出现溢出的问题。于是进行正则化。函数将每一列的数据进行正则化后直接返回。

3.2.3. 梯度下降

1. 先对输入矩阵做正则化。
2. 添加一列，作为偏执 bias-b 的输入（self.w 也是 n+1 维的）。
3. 73 行 for 循环是迭代次数。记录 loss 并更新权重 self.w。
4. 76 行 for 循环是为了计算$\sum_i (y_i - p_1)\hat{x}_i$。
    a) 77 行获得数据。
    b) 78~79 行得到 p1.

c) 80~81 行获得差距和 loss。

d) 82~83 行得到$\sum_i (y_i - p_1)\hat{\boldsymbol{x}}_i$

e) 84 行更新权重。

5. 87 行规定提前终止标准，两次 loss 之间的差小于 epsilon 即可终止。
6. 返回的 loss_list 是为了之后 loss 的作图。

```
67  def gradient_decrease(self,X,y,alpha=0.01,epoch=1000,epsilon=1e-3):
68      m,n = X.shape
69      X = self.normalization(X)
70      #添加“1”列，作为bias-b
71      X = np.c_[X,np.ones((m,1))]
72      loss_list = []
73      for i in range(epoch):
74          temp = np.zeros((n+1,1))
75          loss = 0
76          for j in range(m):#\partial{l(\beta)}/\partial{\beta}
77              x_j = X[j].reshape(-1,1)
78              output_j = np.dot(x_j.T,self.w)
79              p1_j = 1.0 - self.sigmoid(-output_j)
80              error_j = y[j] - p1_j
81              loss += error_j**2
82              temp_j = x_j * error_j
83              temp += temp_j
84          self.w = self.w + alpha*temp
85          loss_list.append(float(loss))
86          #提前停止标准，两次loss之差小于1e-4
87          if(i>=2 and abs(loss_list[i]-loss_list[i-1])<epsilon):
88              break
89      return loss_list
```

3.2.4. 预测

```
91  def fit(self,X):
92      m, n = X.shape
93      X = self.normalization(X)
94      X = np.c_[X, np.ones((m, 1))]
95      res_list = []
96      for j in range(m):
97          #计算输出结果并打印
98          x_j = X[j].reshape(-1, 1)
99          output_j = np.dot(x_j.T, self.w)
100         p1_j = 1.0 - self.sigmoid(-output_j)
101         if(p1_j > 0.5):
102             print('B')
103             res_list.append(1)
104         else:
105             print('M')
106             res_list.append(0)
107     return res_list
```

1. 92~94 初始化预测矩阵。
2. 96 的 for 循环，对每一条数据进行预测：

a) 98~100 计算 p1_j

b) 101~106 根据 p_1 的值输出对应的类别，并记录。

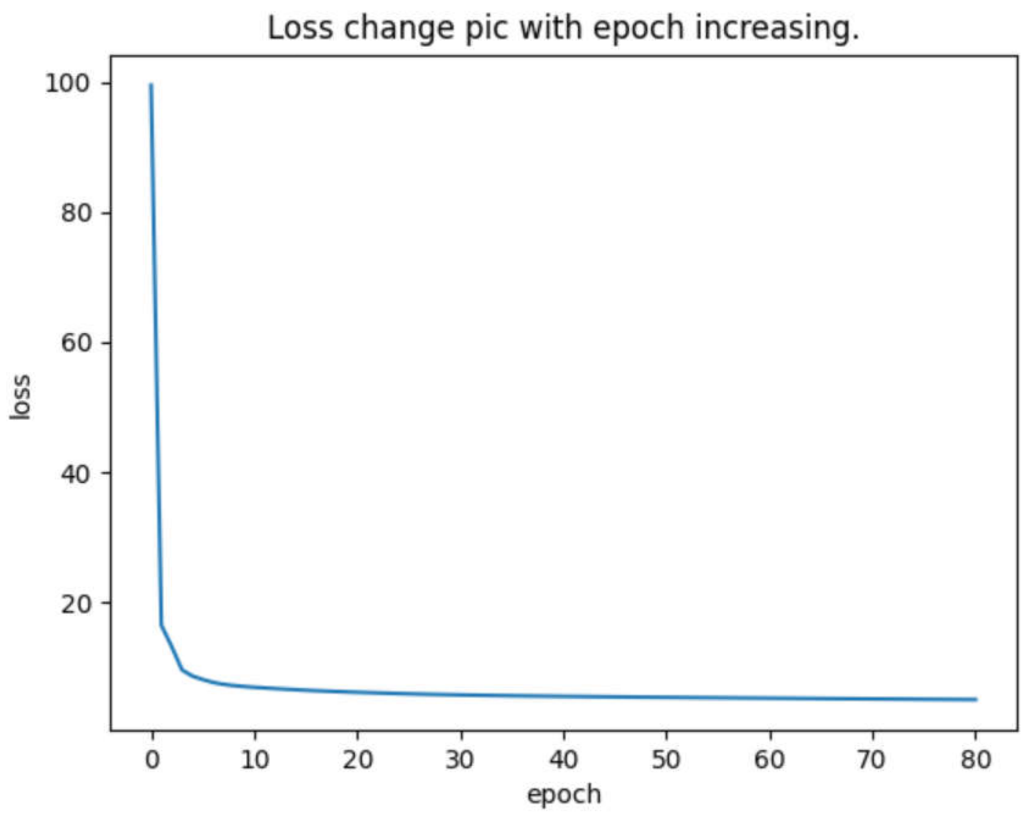
3.3. 主函数

```
110 lr = LogisticRegression(X.shape[1])
111 loss_list = lr.gradient_decrease(X,y)
112 pred = lr.fit(X_test)
```

1. 110 行初始化 LogisticRegression 类。
2. 111 行梯度下降，求得参数 self.w。
3. 112 行对数据进行预测。

4. 实验结果展示

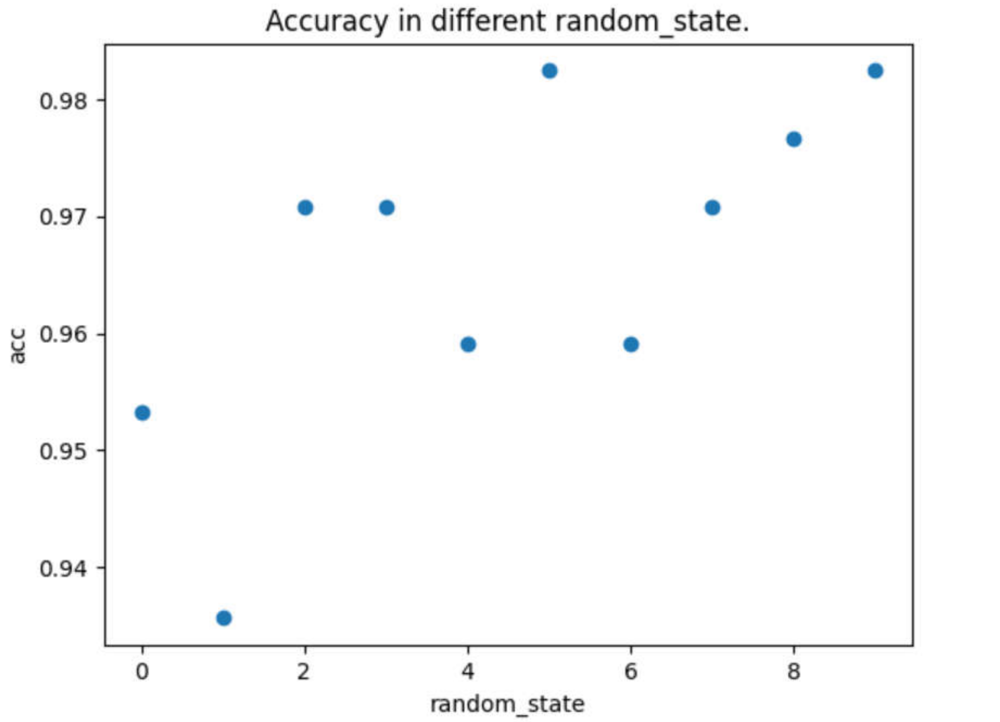
4.1. Loss 的变化



4.2. Acc 的值

采用 sklearn 中的随机划分数据集，设定不同的 random_state 值的方式对预测准确率进行了测试。

在 epsilon=1e-2 时，有如下图



可以看出，预测结果的准确度还是很高的，都在 93%以上。达到了实验要求的精度。