实验报告

howlinggggg

2022 年 10 月 16 日

1　实验要求

1. 编写自己的 Logistic Regression
2. 完成对数据确实项的处理
3. 训练模型并画出 loss 曲线
4. 使用测试集验证模型

2　实验原理

线性模型

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

广义线性模型

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

其中 $g(\cdot)$ 为链接函数，单调可微

线性模型用应用于回归问题：一元/多元线性回归

最小化均方误差

$$\hat{\boldsymbol{w}}^* = \arg\min_{\hat{\boldsymbol{w}}} \|\boldsymbol{y} - \boldsymbol{x}\hat{\boldsymbol{w}}\|_2^2$$

线性模型

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

其中 $y \in \{0, 1\}$

3 实现

Logistic.py

定义了 class LogisticRegression

___init___

初始化 class 参数

```
def __init__(self, lr, iteration, loss, epsilon=0.0001, w=[],
    max=[], min=[], tr_times=0):
   self.lr = lr
   self.iteration = iteration
   self.epsilon = epsilon
   self.w = w
   self.max = max
   self.min = min
   self.loss = loss
   self.tr_times = tr_times
```

其中

- lr: 学习率
- iteration: 学习次数
- loss：存储 loss 参数
- epsilon：梯度下降的阈值
- w：权重矩阵
- max：存储各类别的最大值
- min：存储各类别的最小值

sigmoid function

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

```
def sigmoid(self, z):
    return 1.0/(1.0 + np.exp(-z))
```

grad

计算梯度

```
def grad(self, w, x, y):
    return ((y - self.sigmoid(x @ w)).T @ x).T
```

fit

1. 首先将权重矩阵 w 初始化为 $(d+1)\times 1$，所有值为 1 的向量
2. 求每个类别的最大最小值，储存到 max[]，min[]
3. 将数据归一化处理
4. 将数据从 pandas.dataframe 类型转为 numpy.array
5. 在 x 后增加一列 1 的向量
6. 计算 loss：$\ell(\boldsymbol{w}) = \sum_{i=1}^{m}\left(\log(1+e^{w^\top x_i}) - y_i w^\top x_i\right)$
7. 比较 loss 变化，若小于阈值且超过迭代次数则停止优化

由于在梯度下降后面步长太大不容易找到最优解，设定每次训练将学习率 ×0.95

```
def fit(self, train_x, train_y):

    m = train_x.shape[0]
    d = train_x.shape[1]

    w = np.ones((d + 1, 1))

    for i in range(d):
        self.max.append(train_x.iloc[:, i].max())
        self.min.append(train_x.iloc[:, i].min())
        train_x.iloc[:, i] = (
            train_x.iloc[:, i] - self.min[i]) / (self.max[i] -
                self.min[i])
```

```
train_x = np.array(train_x)
train_x = np.c_[train_x, np.ones(shape=(m, 1))]

train_y = np.array(train_y).reshape(len(train_y), 1)

l1 = 0
for i in range(m):
    l1 += np.log2(1 + np.exp((np.dot(train_x[i], w)[0]))
            ) - train_y[i] * (np.dot(train_x[i], w)[0])

counter = 0

while True:
    counter += 1
    dl = self.grad(w, train_x, train_y)
    w = w + self.lr * dl

    self.lr = 0.95 * self.lr

    l2 = 0
    for i in range(m):
        l2 += np.log2(1 + np.exp((np.dot(train_x[i], w)[0]))) -
            train_y[i] * (
          np.dot(train_x[i], w)[0])

    self.loss.append(l2/m)
    print(counter, l2, len(self.loss))

    if abs(l2-l1) < self.epsilon and counter >= self.iteration:
```

```
            break

        l1 = l2

    self.w = w
    self.tr_times = counter

    print('train', counter, ' times')
```

predict

```
def predict(self, test_x):
    for i in range(test_x.shape[1]):
        test_x.iloc[:, i] = (test_x.iloc[:, i] -
                        self.min[i]) / (self.max[i] - self.min[i])
    test_x = np.array(test_x)
    test_x = np.c_[test_x, np.ones(test_x.shape[0])]

    pre = self.sigmoid(test_x @ self.w).flatten().tolist()
    for i in range(len(pre)):
        if pre[i] > 0.5:
            pre[i] = 1
        else:
            pre[i] = 0
    return pre
```

evaluate

计算准确率

```python
def evaluate(self, pre, test_y):
    test_y = np.array(test_y)

    assert len(pre) == len(test_y)

    counter = 0

    for i in range(len(pre)):
        if pre[i] == test_y[i]:
            counter += 1

    print('correct rate:', counter / len(pre))
```

Loan.py

载入 pandas 和 numpy 并读取数据

```python
import pandas as pd
import numpy as np
df = pd.read_csv('loan.csv')
df.head()
```

encode

将数据编码，并将缺失的数据以均值替代

```python
df.Gender = df.Gender.map({'Male': 1, 'Female': 0})
df.Married = df.Married.map({'Yes': 1, 'No': 0})
df.Dependents = df.Dependents.map({'0': 0, '1': 1, '2': 2, '3+':
```

```
    3})
df.Education = df.Education.map({'Graduate': 1, 'Not Graduate':
    0})
df.Self_Employed = df.Self_Employed.map({'Yes': 1, 'No': 0})
df.Property_Area = df.Property_Area.map(
   {'Urban': 1, 'Semiurban': 0.5, 'Rural': 0})
df = df.fillna({'Gender': 0.5, 'Married': 0.5,
           'Dependents': df['Dependents'].mean(),
           'Self_Employed': df['Self_Employed'].mean(),
           'LoanAmount': df['LoanAmount'].mean(),
           'Loan_Amount_Term': df['Loan_Amount_Term'].mean(),
           'Credit_History': df['Credit_History'].mean()})

df.Loan_Status = df.Loan_Status.map({'Y': 1, 'N': 0})
```

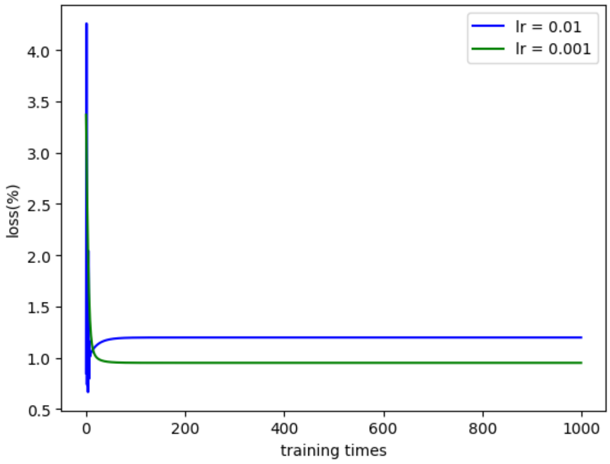
data process

以 9:1 的比例划分训练集和测试集

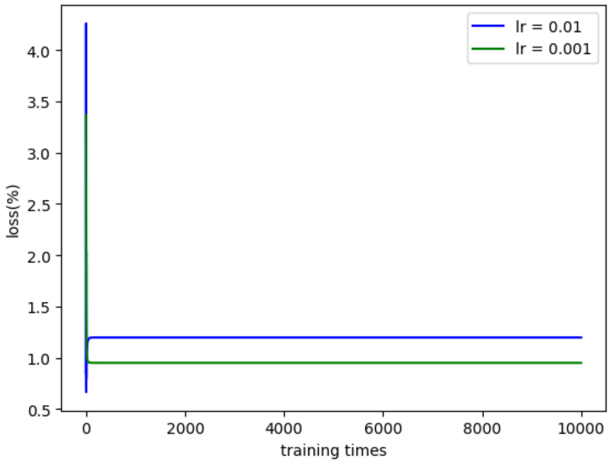
```
train = df.sample(frac=0.9, random_state=3, axis=0)
test = df[~df.index.isin(train.index)]

X_train = train.loc[:, 'Gender':'Loan_Status']
Y_train = train.loc[:, 'Loan_Status':'Loan_Status']
X_test = test.loc[:, 'Gender':'Loan_Status']
Y_test = test.loc[:, 'Loan_Status':'Loan_Status']
```

超参数调整

首先将 lr 分别设定为 0.01 和 0.001 进行比较发现 lr=0.01 时 loss 不降反升，学习率太大反而无法找到最优解

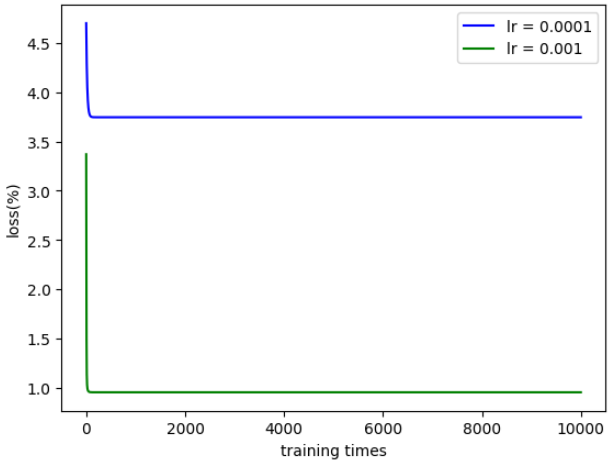


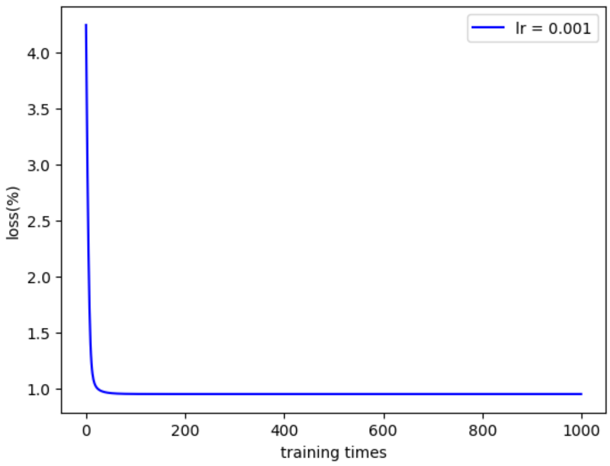
将 iteration 提升到 10000 也不再下降



lr 调整再小也并不会比较好

最终设 lr=0.01 学习到的 w 为 [[-0.27971592] [0.03282886] [0.4653684] [-0.15817547] [0.63212212] [0.88765058] [0.91801593] [0.60895457] [-0.11652041] [0.38934097] [0.17718697] [1.81356296] [-0.61681729]]


lr = 0.0001
lr = 0.001
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
loss(%)
0
2000
4000
6000
8000
10000
training times


lr = 0.001
4.0
3.5
3.0
2.5
2.0
1.5
1.0
loss(%)
0
200
400
600
800
1000
training times

由 w 可知 Loan_Status 和 Self_Employed,Credit_History 较为相关