Machine Learning Lab1

Xiaoma

2022.10.6

实验要求

1. 编写自己的 Logistic Regression class
2. 完成对数据集缺失项的处理，以及数据集的训练预处理
3. 训练模型并画出 loss 曲线
4. 使用测试集验证模型，比较采用不同处理方法的精确度

   实验所使用的数据集

重新设计了类的接口以及其他的处理方法，灵感来源于曾经在课外完成的cs231n-assignment1

实验原理

- 线性模型
  - $f(x) = w^{\mathbf{T}}x + b \quad s.t. \quad f(x) \approx y$
  - w 为权重; b 为偏置; y 为标签
- 广义线性模型

  - $f(x) = g^{-1}(w^{\mathbf{T}}x + b) \quad s.t. \quad f(x) \approx y$
  - $g(\cdot)$ 为链接函数，单调可微

- 线性模型应用于回归问题
  - 最小化均方误差 $\hat{w}^{\star} = arg\min_{\hat{w}} \|y - \mathbf{X}\hat{w}\|_2^2$
  - 用最小二乘法求闭式解 $f(\hat{x_i}) = x_i(\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^T y$

- 本次实验关注线性模型在分类问题的典型应用
  - 逻辑回归

- 逻辑回归
  - 基本形式 $f(x) = \frac{1}{1+e^{-w^{\mathbf{T}}x+b}} \quad s.t. \quad f(x) \approx y$
  - 其中 $y \in 0, 1$，即二分类模型

- 常用的最小化均方误差方法 (本次实验采用梯度下降法)
  - 牛顿法
  - 梯度下降法

实现细节

Logistic.py

class LogicticRegression

该类包含了逻辑回归所需的所有函数 (sigmoid, fit, predict 等)。其中声明了若干变量用于传递参数。

- learning_ rate：学习率
- iterations：迭代器
- w：权重矩阵

- mean：同类参数均值
- var：同类参数方差

```
class LogisticRegression:
'''
learning_rate:学习率
iterations:迭代器
w:权重矩阵
mean:均值
var:方差
'''
def __init__(self, learning_rate, iterations):
    self.learning_rate = learning_rate
    self.interations = interations
    self.w = []
    self.mean = []
    self.var = []
```

sigmoid

实现 Sigmoid 函数

```
#计算sigmoid函数
def sigmoid(self, x):
    return 1.0 / (1 + np.exp(-x))
```

grad

计算梯度 grad

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

fit

传入参数为

- train_x：训练数据参数
- train_y：训练数据标签
- loss：记录 loss 变化

实现步骤

- 首先对 tain_x 进行批量归一化
- 其次使用公式 $l(w) = \sum_{i=1}^{m}(-y_i w^{\mathbf{T}} x_i + \log(1 + e^{w^{\mathbf{T}} x_i}))$ 计算 loss
- 计算 loss 的梯度 $\nabla l(\boldsymbol{w})$
- 首先比较前后两次梯度下降后的 loss 变化，若小于阈值则停止优化，否则继续进行优化
- 最后保存 w 用以分析

***** 考虑到梯度下降的特点，采用了逐步衰减学习率的方法 *****

```
    def fit(self, train_x, train_y, loss):
        w = np.ones((train_x.shape[1] + 1, 1))
        for i in range(train_x.shape[1]):
            self.mean.append(np.mean(train_x.iloc[:, i]))
            self.var.append(np.var(train_x.iloc[:, i]))
            train_x.iloc[:, i] = (train_x.iloc[:, i] - self.mean[i]) / np.sqrt(self.var[i])

        train_x = np.c_[train_x, np.ones(train_x.shape[0])]

        loss_history = 0
        iteration = 0

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

        while iteration < self.iterations:
            iteration += 100
            dl = self.grad(w, train_x, train_y)

            w = w + dl * self.learning_rate * 0.95
            loss_new = 0

            for i in range(train_x.shape[0]):
                loss_new += -train_y[i] * (np.dot(w.T, train_x[i])) + np.log2(1 + np.exp(np.dot(w.T, train_x[i])))
            loss.append(loss_new)
            if abs(loss_history - loss_new) < 0.00001:
                print("***loss is ok***")
                break
            loss_history = loss_new
            print(str(iteration) + "/" + str(self.iterations))
            print("loss :" + str(loss_new/train_x.shape[0]))

        self.w = w
```

predict

根据训练得到的 w 矩阵对测试数据进行预测

```
    def predict(self, test_x):
        for i in range(test_x.shape[1]):
            test_x.iloc[:, i] = (test_x.iloc[:, i] - self.mean[i]) / np.sqrt(self.var[i])

        test_x = np.c_[test_x, np.ones(test_x.shape[0])]

        p = self.sigmoid(test_x @ self.w).T
        p = p.flatten()
        #print(p.shape)
        credit = test_x[:,9]
        for i in range(p.shape[0]):
            if p[i] > 0.5:
                p[i] = 1
            elif credit[i] == 1:
                p[i] = 1
            else:
                p[i] = 0
        return p
```

Loan.ipynb

数据读取以及引入必要的库

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

打印数据集信息

```
df.info()
```

丢弃无用的参数

```
df.drop("Loan_ID", axis=1, inplace=True)
```

数据处理（旧方法）

数据处理过程

- 对于 Gender，将女性参数转换为 0，男性转换为 0，缺失数据转换为 0.5
- 对于 Graduate，将 Graduate 参数转换为 1，Not Graduate 转换为 0
- 对于 Married，将已婚参数转换为 0，未婚参数转换为 1，缺失数据转换为 0.5
- 对于 Self_Employed，将 Yes 转换为 0，No 转换为 1，缺失数据转换为 0.5
- 对于 LoanAmount，将缺失数据转换为参数的平均值
- 对于 Loan_Status，将 Y 转换为 1，N 转换为 0

数据处理（优化方法）

对单一类型的参数与结果之间的关系进行分析

Gender

将男性申请成功与失败的比例和女性申请成功和失败的比例进行对比

```
male_yes =  len(df1[(df1['Loan_Status'] == 'Y') & (df1['Gender'] == 'Male')  ])
male_no =   len(df1[(df1['Loan_Status'] == 'N') & (df1['Gender'] == 'Male')  ])
female_yes =len(df1[(df1['Loan_Status'] == 'Y') & (df1['Gender'] == 'Female')])
female_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Gender'] == 'Female')])
male_yes, male_no, female_yes, female_no ,male_yes/male_no, female_yes/female_no

output:(339, 150, 75, 37, 2.26, 2.027027027027027)
```

发现男性申请成功的概率要大于女性，故将性别参数替换

```
male_replace = male_yes/male_no / (male_yes/male_no + female_yes/female_no)
female_replace = female_yes/female_no / (male_yes/male_no + female_yes/female_no)
gender_n_repalce = (male_replace + female_replace)/2
male_replace,female_replace, gender_n_repalce

output:(0.5271718572689446, 0.4728281427310554, 0.5)
```

Graduate

比较 Graduate/Not Graduate 与申请成功率的关系

```
graduate_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Education'] == 'Graduate')  ])
graduate_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Education'] == 'Graduate')  ])
ngraduate_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Education'] == 'Not␣Graduate')  ])
ngraduate_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Education'] == 'Not␣Graduate')  ])
graduate_yes, graduate_no, ngraduate_yes, ngraduate_no, graduate_yes/graduate_no,ngraduate_yes/ngraduate_no

output:(340, 140, 82, 52, 2.4285714285714284, 1.5769230769230769)
```

发现 Graduate 且申请成功的概率大于 Not Graduate 且申请成功的概率，故替换参数

```
graduate_replace =  graduate_yes/graduate_no / (graduate_yes/graduate_no + ngraduate_yes/ngraduate_no)
ngraduate_replace = ngraduate_yes/ngraduate_no / (graduate_yes/graduate_no + ngraduate_yes/ngraduate_no)
graduate_replace, ngraduate_replace

```

```
    output:(0.6063100137174211, 0.39368998628257884)
```

Married

比较 Married 与申请成功率的关系

```
    married_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Married'] == 'Yes')  ])
    married_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Married'] == 'Yes')  ])
    nmarried_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Married'] == 'No')  ])
    nmarried_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Married'] == 'No')  ])
    married_yes,married_no,nmarried_yes,nmarried_no,married_yes/married_no,nmarried_yes/nmarried_no

output:(285, 113, 134, 79, 2.52212389380531, 1.6962025316455696)
```

发现 Married 且申请成功的概率大于 Not Married 且申请成功的概率，故替换参数

```
    married_replace = married_yes/married_no / (married_yes/married_no + nmarried_yes/nmarried_no)
    nmarried_replace = nmarried_yes/nmarried_no / (married_yes/married_no + nmarried_yes/nmarried_no)
    married_n_replace = (married_replace+nmarried_replace)/2.0
    married_replace,nmarried_replace,married_n_replace

    output:(0.5978968053748308, 0.4021031946251693, 0.5)
```

Self_Employed

比较 Self_Employed 与申请成功率的关系

```
    self_employed_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Self_Employed'] == 'Yes')  ])
    self_employed_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Self_Employed'] == 'Yes')  ])
    nself_employed_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Self_Employed'] == 'No')  ])
    nself_employed_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Self_Employed'] == 'No')  ])
    self_employed_yes,self_employed_no,nself_employed_yes,nself_employed_no,self_employed_yes/self_employed_no,nself_employed_yes/nself_employed_no

output:(56, 26, 343, 157, 2.1538461538461537, 2.1847133757961785)
```

发现二者申请成功的概率没有显著差别

Property_Area

比较 Property_Area 与申请成功率的关系

```
    urban_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Property_Area'] == 'Urban')  ])
    urban_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Property_Area'] == 'Urban')  ])
    semiurban_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Property_Area'] == 'Semiurban')  ])
    semiurban_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Property_Area'] == 'Semiurban')  ])
    rural_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Property_Area'] == 'Rural')  ])
    rural_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Property_Area'] == 'Rural')  ])
    urban_yes,urban_no,semiurban_yes,semiurban_no,rural_yes,rural_no  ,urban_yes/urban_no,semiurban_yes/semiurban_no,rural_yes/rural_no

output:(133,69,179,54,110,69,1.9275362318840579,
            3.3148148148148148 15,1.5942028985507246)
```

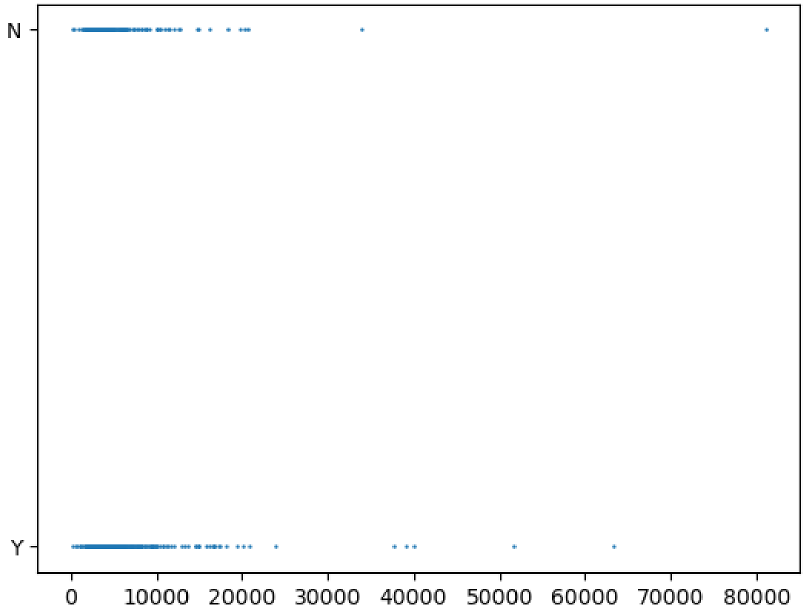
发现三者申请成功的概率具有较大差异，故替换参数

```
urban_replace = urban_yes/urban_no/(urban_yes/urban_no+semiurban_yes/semiurban_no+rural_yes/rural_no)
semiurban_replace = semiurban_yes/semiurban_no/(urban_yes/urban_no+semiurban_yes/semiurban_no+rural_yes/rural_no)
rural_repalce = rural_yes/rural_no/(urban_yes/urban_no+semiurban_yes/semiurban_no+rural_yes/rural_no)
urban_replace,semiurban_replace,rural_repalce

output:(0.28194558944765047, 0.48486632905429283, 0.23318808149805678)
```

ApplicantIncome

比较 ApplicantIncome 与申请成功率的关系

```
x = df1[”ApplicantIncome”]
y = df1[”Loan_Status”]
plt.scatter(x, y,s=0.5)
```

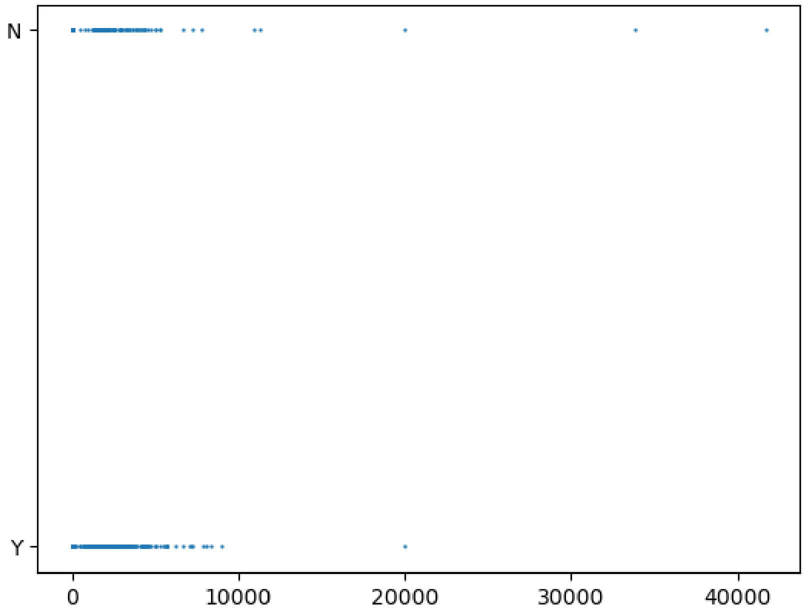


无法得出 ApplicantIncome 高低与申请成功的差异，故不做额外处理

CoapplicantIncome

比较 CoapplicantIncome 与申请成功率的关系

```
    x = df1["CoapplicantIncome"]
    y = df1["Loan_Status"]
    plt.scatter(x, y,s=0.5)
```

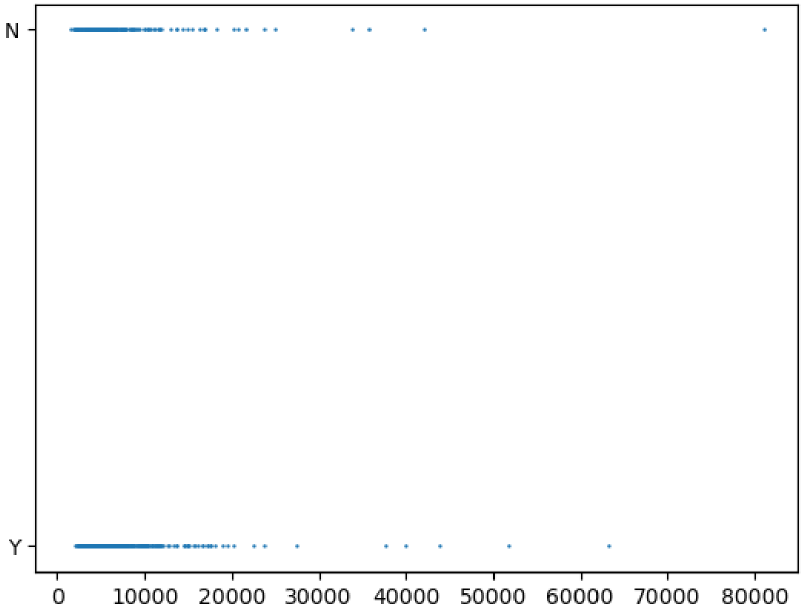


无法得出 ApplicantIncome 高低与申请成功的差异，故不做额外处理

ApplicantIncome+CoapplicantIncome

将二者求和进行分析

```
x = df1["CoapplicantIncome"]+df1["ApplicantIncome"]
y = df1["Loan_Status"]
plt.scatter(x, y,s=0.5)
```



无法得出 ApplicantIncome+CoapplicantIncome 高低与申请成功的差异，故不做额外处理

Credit_History

比较 Credit_History 与申请成功率的关系

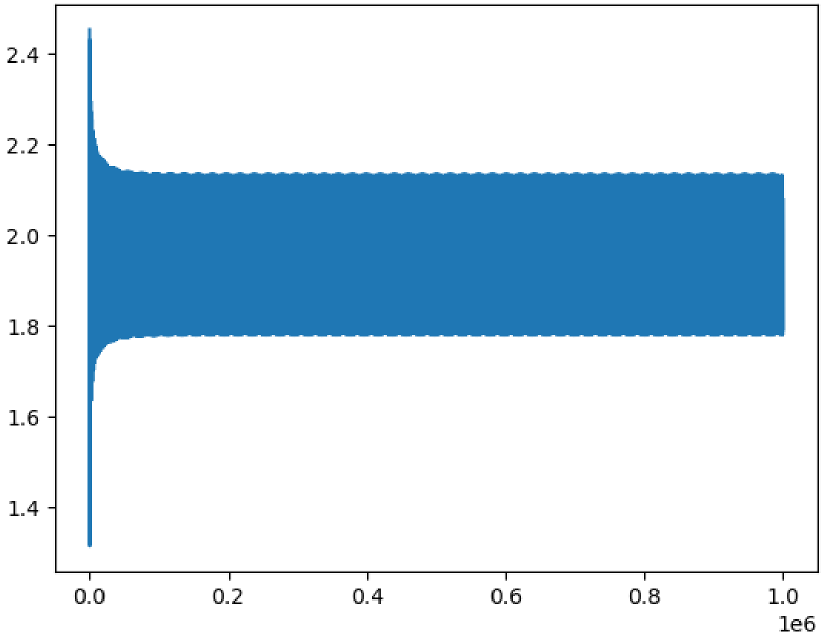
```
credit_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Credit_History'] == 1)  ])
credit_no = len(df1[(df1['Loan_Status'] == 'N') & (df1['Property_Area'] == 1)  ])
ncredit_yes = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Credit_History'] == 0)  ])
ncredit_no = len(df1[(df1['Loan_Status'] == 'Y') & (df1['Credit_History'] == 0)  ])
credit_yes,credit_no,ncredit_yes,ncredit_no,ncredit_yes/ncredit_no

output:(378, 97, 7, 7, 3.8969072164948453, 1.0)
```

发现二者申请成功的概率具有较大差异，故替换参数

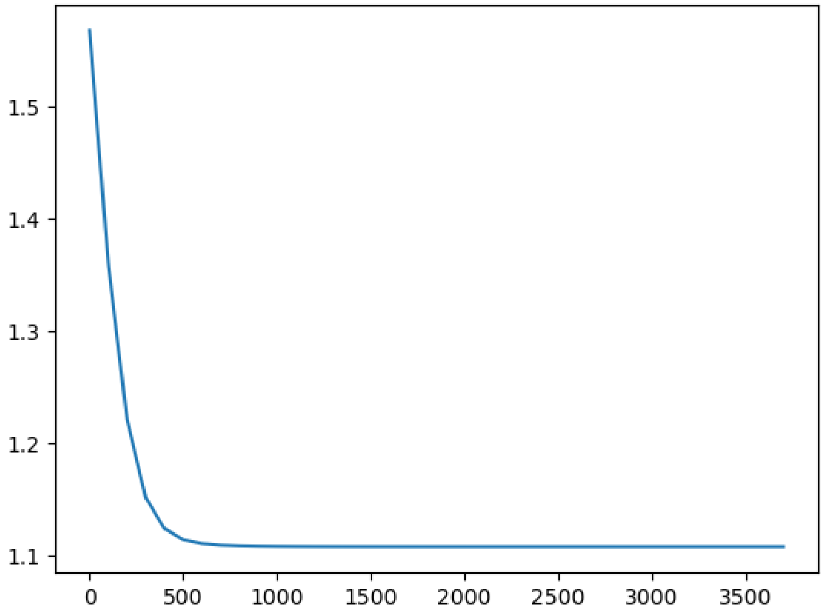
```
credit_replace = credit_yes/credit_no/(credit_yes/credit_no+
    ncredit_yes/ncredit_no)
ncredit_replace = ncredit_yes/ncredit_no/(credit_yes/
    credit_no+ncredit_yes/ncredit_no)
credit_replace,ncredit_replace

output:(0.7957894736842105, 0.20421052631578945)
```

超参数调整过程

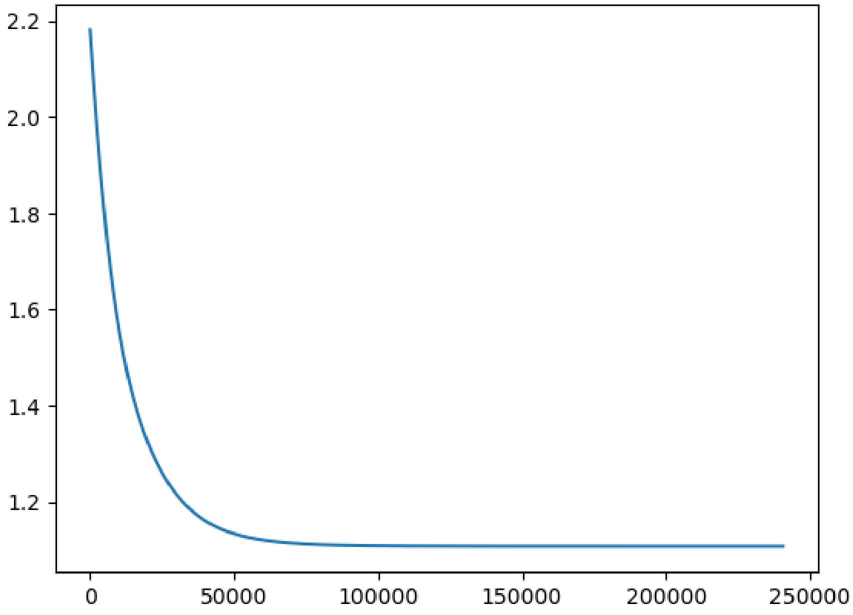
初始设置 learning_rate：0.03，iterations：10000000
首次训练 loss 曲线如图所示, 发现 loss 曲线发生较大波动，需要降低学习率。



继续设置 learning_rate：0.01，iterations：10000000
loss 曲线如图所示，发现 loss 曲线虽逐步下降但仍不够平滑，继续调节学习率。



多次调节以后（比较了多次训练的 loss 曲线）
设置 learning_rate：0.0001，iterations：10000000
loss 曲线如图所示，曲线平滑且稳定下降，故作为训练参数。

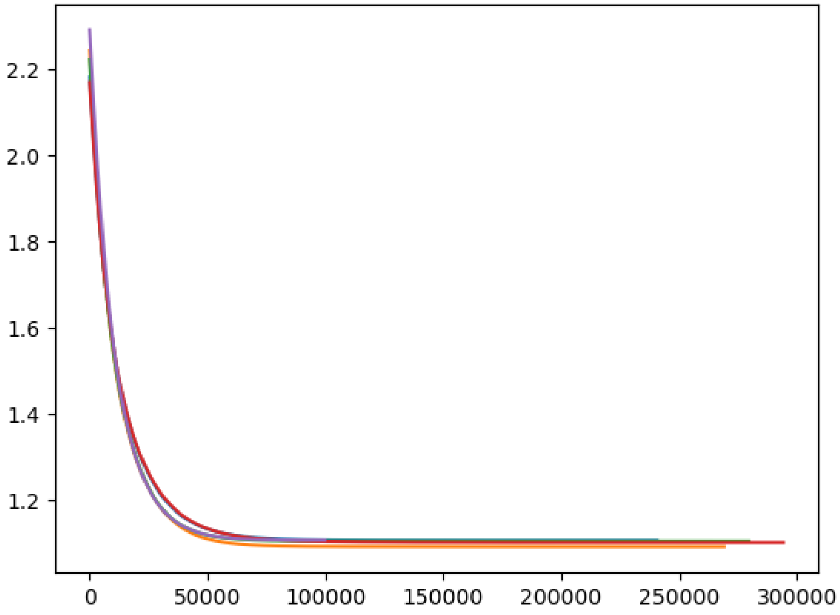


精确度分析 (五折交叉验证)

learning_rate： 0.0001, iterations： 10000000

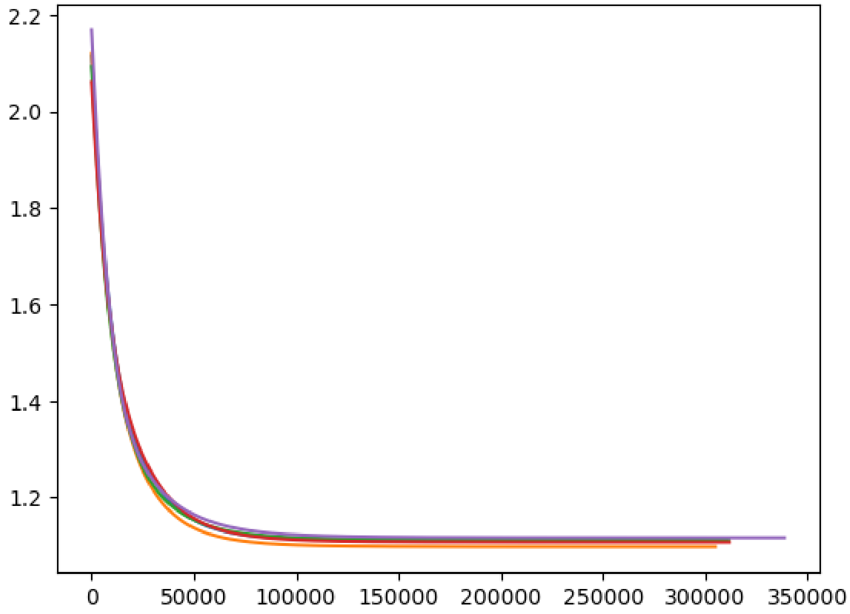
旧方法

loss 曲线



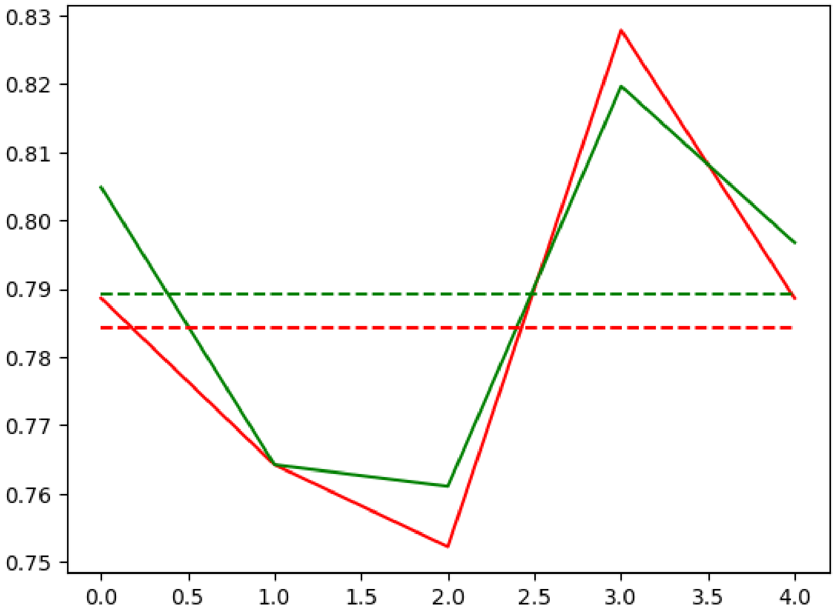
优化方法

loss 曲线



精确度比较

其中红色折线为旧方法精确度，绿色折线为优化方法精确度



旧方法平均精确度为：**0.7843**, 优化方法平均精确度为：**0.7893**