Lab 1 实验报告

PB20000296 郑滕飞

py 文件：

1、整体框架

由于采用梯度下降法并需要绘制损失曲线，需要计算损失函数及其梯度。而为了作出预测，需要计算 Sigmoid 函数。这三个函数直接根据 ppt 提供的公式实现即可：

[此处计算损失函数及其梯度是按照 ppt 的公式，并没有对样本数归一化，具体对比见第三部分]

```python
def sigmoid(self, x):
    """The logistic sigmoid function"""
    return 1.0 / (1 + np.exp(-np.dot(self.w, x)) )
```

```python
def count_loss(self, X, y):
    """count loss"""
    loss = 0
    for i in range(y.size):
        loss -= y[i] * np.dot(self.w, X[i])
        loss += np.log(1 + np.exp(np.dot(self.w, X[i])))
```

```python
def count_grad_loss(self, X, y):
    """count gradient of loss"""
    grdloss = np.zeros(X[0].size)
    for i in range(y.size):
        grdloss -= (y[i] - 1 + 1.0/(1 + np.exp(np.dot(self.w, X[i]))))*X[i]
```

对于学习的部分，需要先考虑 fit 的写法。注意到，匹配截距相当于给 X 增加一列全为 1 的列，因此这个操作可直接于最开始进行，之后再根据当前 X 的列数建立 w 即可。如果匹配截距，w 的最后一个分量即为截距：

```python
if self.fit_intercept == True:
    X = np.c_[X, np.ones(y.size)]
self.w = np.zeros(X[0].size)
```

梯度下降的过程则直接按照算法进行：

```python
while np.sum(self.count_grad_loss(X, y) ** 2) > tol:
    if count == max_iter:
        print("reached max iter")
        break
    count += 1
    ls.append(self.count_loss(X, y))
    self.w = self.w - lr * self.count_grad_loss(X, y)
```

由于在 count_loss 中计算出了损失的值，直接将这些值储存成向量 ls 返回，即可绘制损失曲线。

lr 用于控制学习率，也即梯度下降的速度，而 tol 和 max_iter 从两个不同方向确定循环的界。在初值设定 tolerance 1e-4 与 max_iter 1e3 时，一般是由于到达最大迭代次数而返回。

预测部分，直接通过 Sigmoid 函数生成预测值：

```
if self.fit_intercept == True:
    X = np.c_[X, np.ones(X.shape[0])]
y_p = []
for i in range(X.shape[0]):
    if (self.sigmoid(X[i]) > 0.5):
        y_p.append(1)
    else:
        y_p.append(0)
```

2、正则化

刚才的过程中，并没有考虑 penalty 参数与 gamma 参数形成的正则化项。通过查阅资料发现，penalty 为 l1 相当于在损失中添加一项$\gamma||w||_1$，而为 l2 则相当于添加$\frac{1}{2}\gamma||w||_2^2$。前者的梯度为 gamma 倍的 sgn(w)，其中 sgn 为符号函数，而后者的梯度即为 gamma 倍的 w。于是，通过正则化参数可以得到最终的损失与损失梯度：

```
if self.penalty == "l1":
    return loss + self.gamma * np.sum(np.abs(self.w))
return loss + self.gamma * np.sum(self.w ** 2) / 2.0
```

```
if self.penalty == "l1":
    return self.gamma * np.sign(self.w) + grdloss
return self.gamma * self.w + grdloss
```

可以发现，正则化由于加入了 w 的模长作为最小化条件，控制了 w 的模不会过大，这点在迭代次数升高时尤为明显[见后方参数比较]。

ipynb 文件：

1、Data Cleaning & Encode

由于较多项目有 Null 项，对连续的属性采取平均值填充后再去掉 Null：

```
df["LoanAmount"].fillna(df["LoanAmount"].mean(), inplace=True)
df["Loan_Amount_Term"].fillna(df["Loan_Amount_Term"].mean(), inplace=True)
df.dropna(inplace=True)
```

而编码时，为了之后归一化方便，离散属性的编码均使用 0 到 1 间的实数：

```
df.Gender=df.Gender.map({'Male':1,'Female':0})
df.Education=df.Education.map({'Graduate':1,'Not Graduate':0})
df.Married=df.Married.map({'Yes':1,'No':0})
df.Dependents=df.Dependents.map({'0':0,'1':0.25,'2':0.5,'3+':1})
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.Loan_Status=df.Loan_Status.map({'Y':1,'N':0})
```

对后代数量，此处假设 1、2、3+的比例是 1:2:4。

2、Data Process

这块分为四个部分：导入 numpy 向量/矩阵、归一化、随机打乱与按比例划分训练集/测试集。

导入部分先用行导入列，再进行转置，归一化则需要利用属性的最大值进行：

```
X[0] = np.array(df.Gender)
X[1] = np.array(df.Married)
X[2] = np.array(df.Dependents)
X[3] = np.array(df.Education)
X[4] = np.array(df.Self_Employed)
X[5] = np.array(df.ApplicantIncome)
X[6] = np.array(df.CoapplicantIncome)
X[7] = np.array(df.LoanAmount)
X[8] = np.array(df.Loan_Amount_Term)
X[9] = np.array(df.Credit_History)
X[10] = np.array(df.Property_Area)
X[5] = X[5] / np.max(X[5])
X[6] = X[6] / np.max(X[6])
X[7] = X[7] / np.max(X[7])
X[8] = X[8] / np.max(X[8])
X = X.T
```

由于数据集不大，且正例与反例比例接近 2:1，差别不大，不需要进行过采样/欠采样等。此外，原数据中分布就已经随机排列，事实上不打乱也可以直接学习。不过为了检验效果，仍随机打乱样本，并将前一部分划分为训练集[默认比例 70%]，后一部分为测试集：

```
index = np.random.permutation(y.size)
X = X[index]
y = y[index]
t = (int)(7 * y.size / 10)
X_train = X[0:(t-1),:]
X_test = X[t:(y.size-1),:]
y_train = y[0:(t-1)]
y_test = y[t:(y.size-1)]
print('size: ' + str(y.size))
print('train size: ' + str(t))
```

3、Train & Test

采用 py 文件中写好的类进行训练：

```
LR = LogisticRegression("l2", 0.5, True)
dr = LR.fit(X_train, y_train, 0.005, 1e-7, 1e3)
```

这里的 dr 保存了每次迭代损失所构成的向量，测试后直接用其进行画图：

```
y_p = LR.predict(X_test)
acc = 1 - np.sum(np.abs(y_p - y_test))/y_p.size
```

```
plt.plot(dr)
print("accuracy:")
print(acc)
```

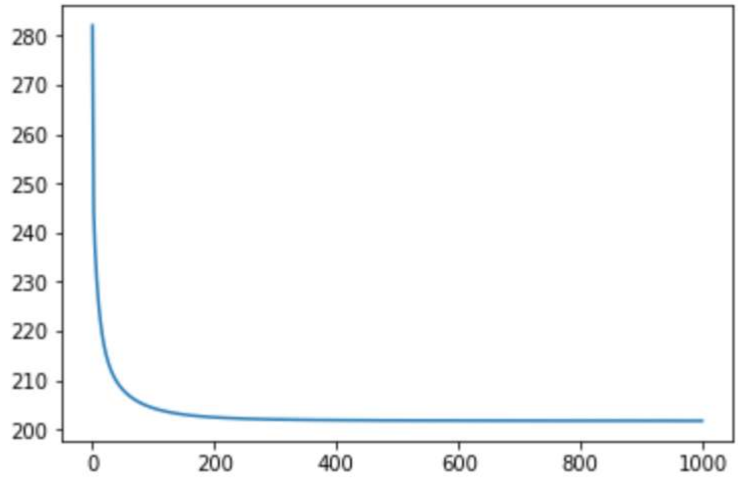
展示与对比：

1、最佳准确率及其损失曲线

数据集分割：数据集大小 511 训练集大小 408(80%) 随机分割

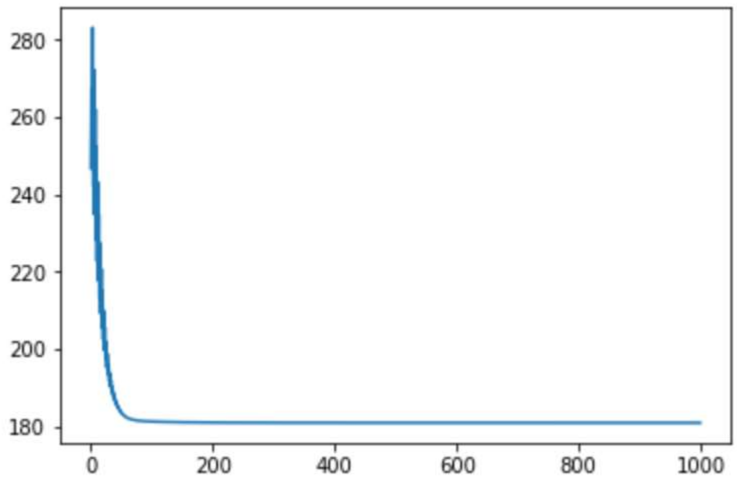
参数：
penalty = ’l2’ gamma = 0 fit_intercept = False
lr = 0.005 tol = 1e-7 max_iter = 1e3

accuracy:
0.8823529411764706



数据集分割：数据集大小 511 训练集大小 357(70%) 随机分割
参数：
penalty = ’l2’ gamma = 0.45 fit_intercept = True
lr = 0.007 tol = 1e-7 max_iter = 1e3

accuracy:
0.8627450980392157



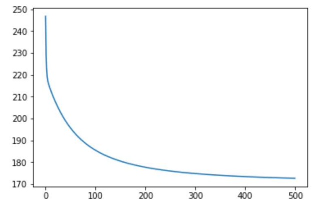
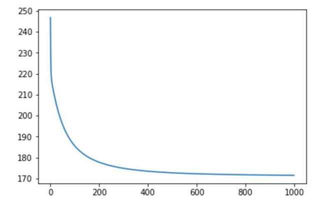
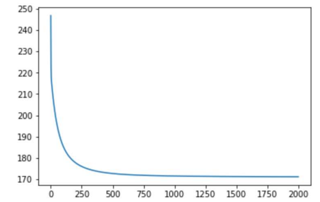
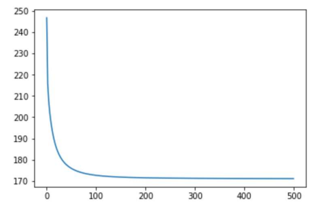
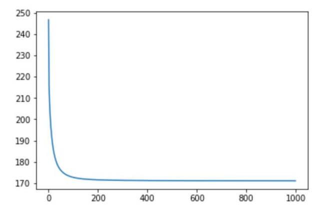
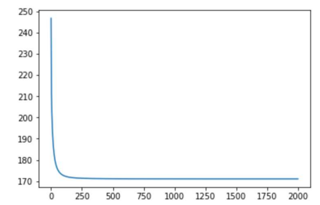
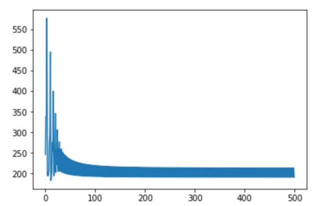
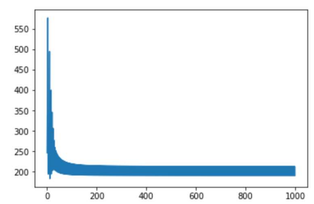
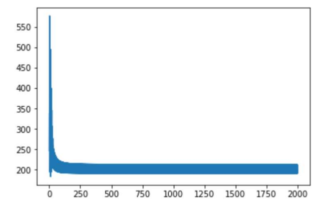
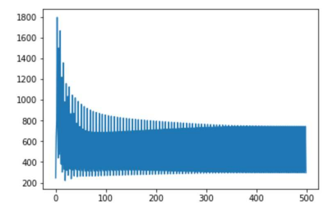
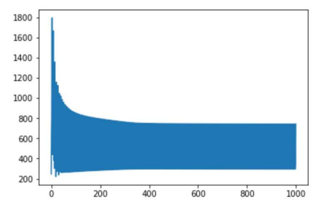
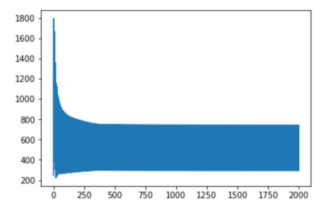
2、参数影响
默认参数为
penalty = ’l2’ gamma = 0 fit_intercept = True
lr = 0.01 tol = 1e-4 max_iter = 1e3
以下叙述参数影响时，未提及的参数均为默认参数。

首先，关于 loss 的归一化问题，根据计算过程不难发现，在计算过程中对 loss 进行归一化，相当于对学习率乘了样本量的倒数，因此，loss 归一化的影响可直接通过学习率进行判别。

其次，由于数据集本身是乱序分布，以下为了对比不同参数造成的影响，去掉了随机打乱的步骤，而是直接以前 70%作为训练集，后 30%作为测试集。

学习率与迭代次数的影响，从直觉上来说，学习率越低则越稳定，但收敛速度会有一定下降，具体的列表如下[表格最左代表 lr，最上代表 max_iter]：

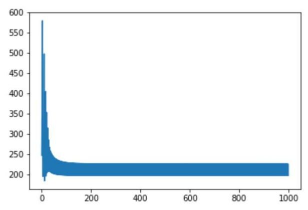
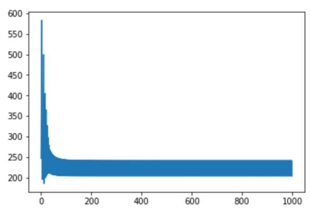
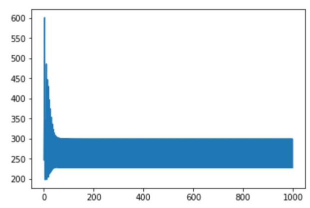
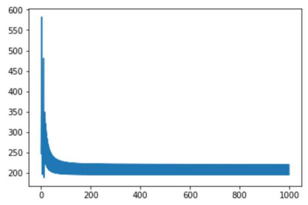
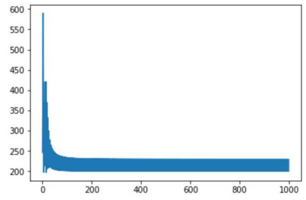
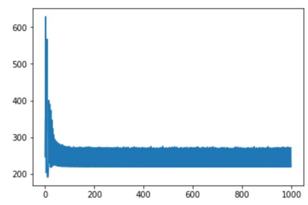
acc	5e2	1e3	2e3
1e-3	0.824	0.824	0.810
5e-3	0.804	0.797	0.797
1e-2	0.824	0.824	0.824
2e-2	0.418	0.817	0.424

loss	5e2	1e3	2e3
1e-3			
5e-3			
1e-2			
2e-2			

可以发现，lr 在 0.005 时曲线都较为稳定，而 0.01 开始已经出现了明显的震荡。当学习率进一步增大时，过拟合也变得更加明显(lr 为 0.02 时迭代次数升高后准确率大幅下降)。

下面观察正则化项的影响[作为参考，上方默认参数时准确率为 0.824]。

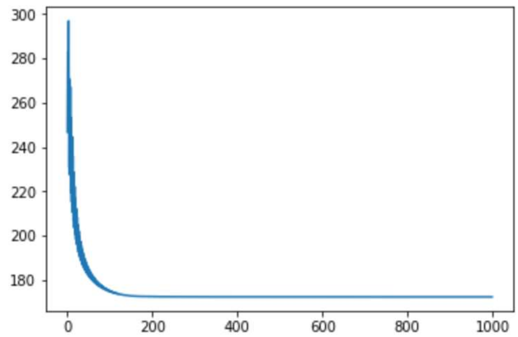
acc	0.2	0.5	2
l1	0.837	0.837	0.778
l2	0.837	0.837	0.837

loss	0.2	0.5	2
l1			
l2			

在正则化项增强时，损失曲线事实上变得更加波动了，这也符合其防止过拟合、保持一定损失的要求。大部分情况下，增加正则化项后结果可以变得更好。不过，当它过强时，由于造成的波动太大，拟合精度事实上降低了。

此外，l2 正则化的效果比 l1 更加明显，这是由于其梯度直接与 w 的值相关，而不是只与符号相关。

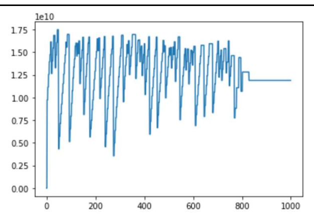
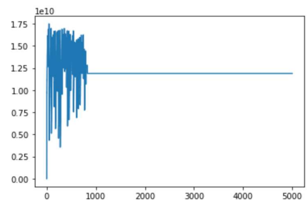
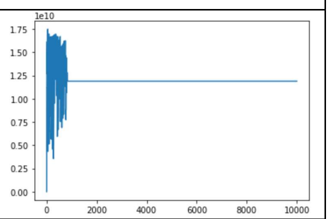
最后，不匹配截距时，收敛事实上变得更加稳定了(这很可能是因为归一化后截距参数造成的影响是很大的)，但准确率下降到了 0.804。



[对于 tol 参数，由于其事实上影响的是迭代次数，可以通过迭代次数的情况观察出它的可能影响。其在实际操作时可以作为一个终点，也有防止过拟合的作用。]

2、数据集操作影响

有关归一化：若不进行归一化，由于数字的量级不同，收敛速度会慢非常多，结果也不尽人意，在其他参数默认时不进行归一化的结果如下：

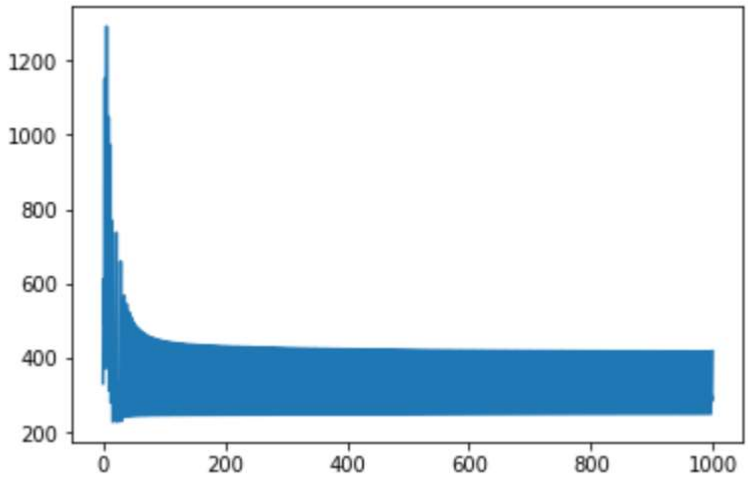
iter	1e3	5e3	1e4
loss			
acc	0.294	0.706	0.307

*由于 loss 计算过程出可能无现穷值，这时需要进行“去头”才能画出曲线：

```
for i in range(len(dr)):
    if dr[i] == float('inf'):
        dr[i] = dr[i-1]
```

正因如此，最终的“稳定”并不是真实的稳定值，而是由于无穷值被填充为了上一个值，此后的 loss 仍然非常大。

有关去掉 NA 值：如果把所有的 Null 全部去掉，样本个数会进一步减少。去掉离散值 Null 后剩余 511 个样本，而去掉全部后只剩下 480 个，学习效果也普遍更差[默认参数准确率 0.811，随机打乱后平均基本小于 80]：



4、算法评价

参数[默认参数加入一定正则化]：

penalty = ’l2’ gamma = 0.3 fit_intercept = True

lr = 0.01 tol = 1e-4 max_iter = 1e3

10 次随机 7:3 留出的结果：

.817 .791 .817 .778 .778 .817 .830 .797 .843 .837

*平均为 81.05%

总结：

虽然乍看实验文档啥也没看明白，探索、写出自己的学习器还是颇有趣的过程。

*调参真的会上瘾啊（恼