---

# 1数据预处理

**代码** `high_frequence_words_5000.py`（用于统计词频并获取独热编码）

- 首先从原始数据集`1998-01-2003版-带音.txt`统计前5000的动词高频词，同时，为了训练效果理性，加入了部分高频副词。（实际上只提取出4999个高频词，剩下一个维度作为未登录词，避免在后面转换为one hot 编码时出现零向量的情况）
- 然后将原始数据集的文本转换为5000维向量存入one_hot.txt文本中。

**代码** `match_X_Y.py`（用于将one_hot.txt与匹配训练集\验证集\测试集并提取词向量）

- 首先根据时间匹配上原始数据集（5000维）和某一个数据集（2维）。
- 然后将匹配上的5000维向量和2维向量写入对应的新的文本中，以便后续数据读入。

# 2逻辑回归模型

## 2.1数据读入

因为txt文本中的5000维与2维词向量为string类型，所以需要先将其处理成矩阵(matrix)/张量(tensor)类型。以5000维为例，将n个向量存入一个list中，最后通过`np.concatenate(list_name)`将其转换为n*5000的矩阵。通过`torch.tensor(matrix_name)`将矩阵转换为张量。

## 2.2构建逻辑回归模型

- **构造预测函数**即sigmoid函数为：

$$h_\theta(X) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

该函数有特殊含义，它表示为结果取1的概率：

$$P(y=1|x;\theta) = h_\theta(x)$$

$$P(y=0|x;\theta) = 1 - h_\theta(x)$$

- **构造损失函数loss**：

$$l(\theta) = \log(L(\theta)) = \sum_{i=1}^{m}[y^{(i)} \log h(\theta)(1|x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(1|X^{(i)}))]$$

- 在**手动求导**过程中采用**批量梯度下降**求解参数$\theta$:

$$\theta := \theta + \alpha(\sum_{i=1}^{m}[y^{(i)} - g(\theta^T x^{(i)})]x^{(i)})$$

- 在**自动求导**过程中使用torch中的**库函数**`torch.optim.Adam`进行优化，Adam算法是对比梯度下降算法更高效的优化算法。
- **评价指标**:

  采用**f1 measure**，计算查准率（precesion）、查全率（recall）

  查准率：预测为正的样本中预测正确的比例

  查全率：数据集中所有正样本被预测为正样本的比例、

$$f1measure = \frac{2 * precesion * recall}{precesion + recall}$$

# 2.3numpy库手动求导

**代码**`LogisticRegression_manual.py`

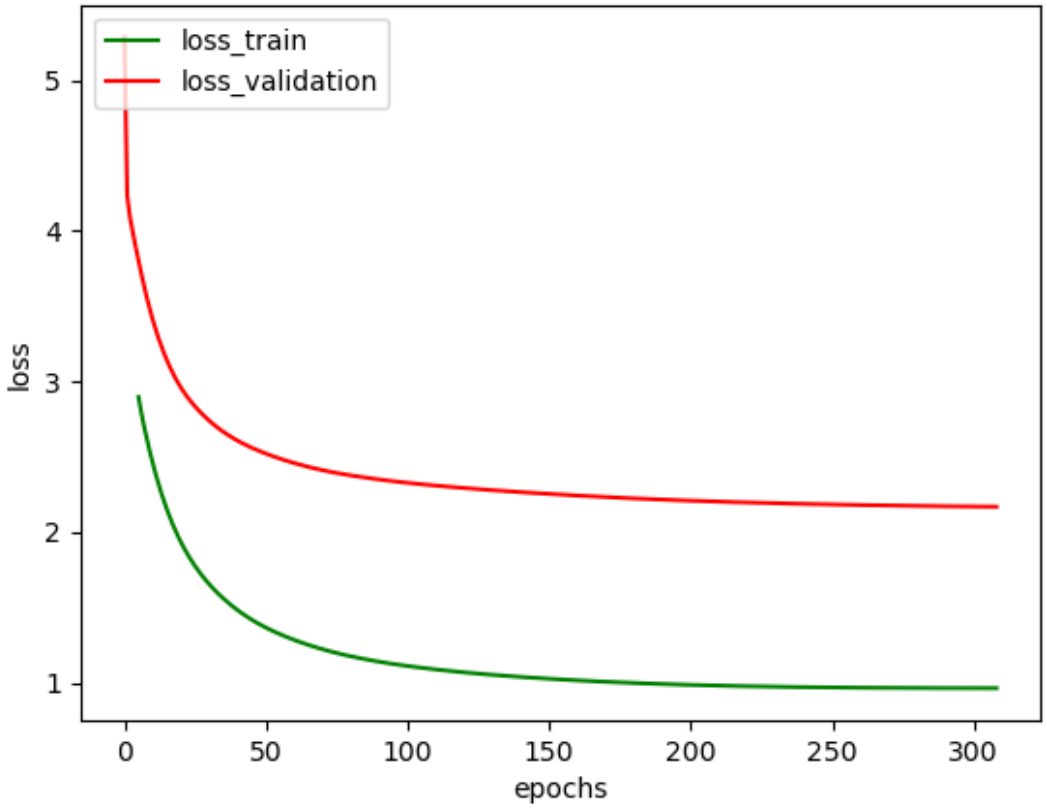
- 确定学习率为10，最大迭代次数为1000，精度为1e-6，当loss函数的值在参数更新前后的差小于精度时终止迭代。
- 在训练集中，通过loss函数计算loss值，通过sigmoid函数计算预测结果为1的概率，再通过梯度下降的方法求解本次迭代中的参数$\theta$，计算其f1 measure
- 在验证集中，通过loss函数计算loss值，通过sigmoid函数带入最新参数$\theta$对验证集中的矩阵进行预测,计算其f1 measure
- 迭代终止后，得到最终参数$\theta$，对测试集进行测试，计算其f1 measure

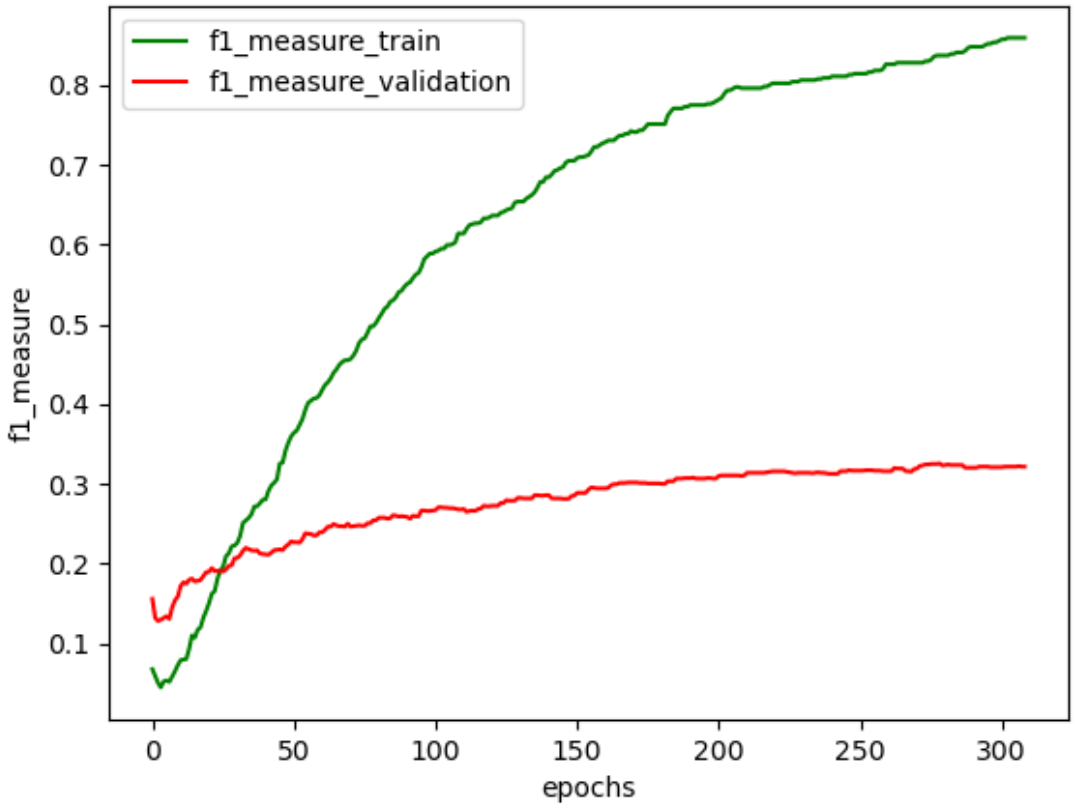
```
epochs:308 loss:0.9656058042270729
train:
precision_ratio=0.9601449275362319 recall_ratio=0.7771260997067448 f1_measure=0.8589951377633711
validation:
precision_ratio=0.32142857142857145 recall_ratio=0.32142857142857145 f1_measure=0.32142857142857145
迭代到第308次，结束迭代!
```

```
test:
precision_ratio=0.38144329896907214 recall_ratio=0.37 f1_measure=0.3756345177664974
test f1_measure:0.3756345177664974
```

根据结果显示，在迭代308次后，训练集的f1 measure大致收敛于0.85，验证集的f1 measure大致收敛于0.32，测试集f1 measure为0.37。

下图为利用`matplotlib`库绘制的loss曲线和f1 measure曲线。可直观看出，在梯度下降的过程中，训练集和测试集的loss函数值逐渐降低，最终趋于稳定，而**f1 measure** 逐渐升高,最终趋于稳定，达到训练目的。




## 2.4pytorch库自动求导

**代码** `LogisticRegression_auto_by_torch`

- 确定学习率为0.01，迭代次数为1000。
- 通过torch库里面的函数`torch.nn.Module`定义模型结构，输入层为5000维，输出层为2维，利用`torch.nn.functional.sigmoid`进行激活。

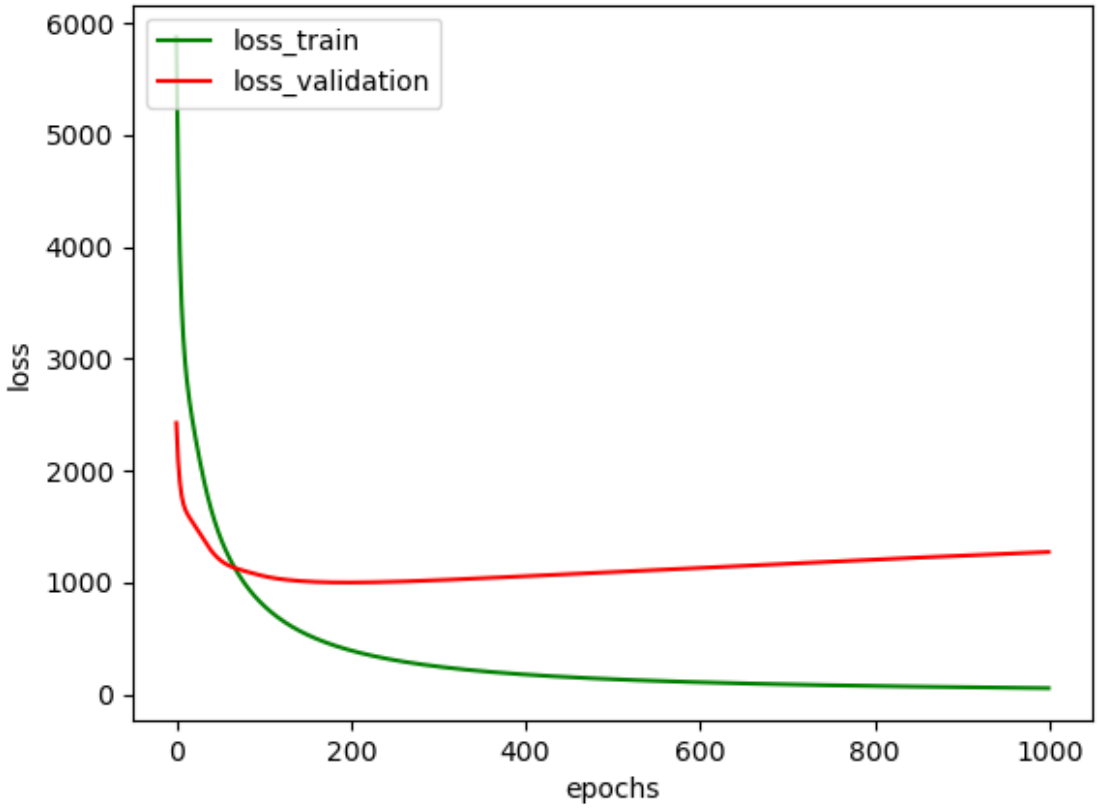
```
class LogisticRegressionModel(torch.nn.Module):
    def __init__(self):
        super(LogisticRegressionModel, self).__init__()
        # 定义模型结构
        self.linear = torch.nn.Linear(5000, 2)  # 输入5000维，输出2维

    def forward(self, x):
        g = torch.nn.functional.sigmoid(self.linear(x))
        return g
```

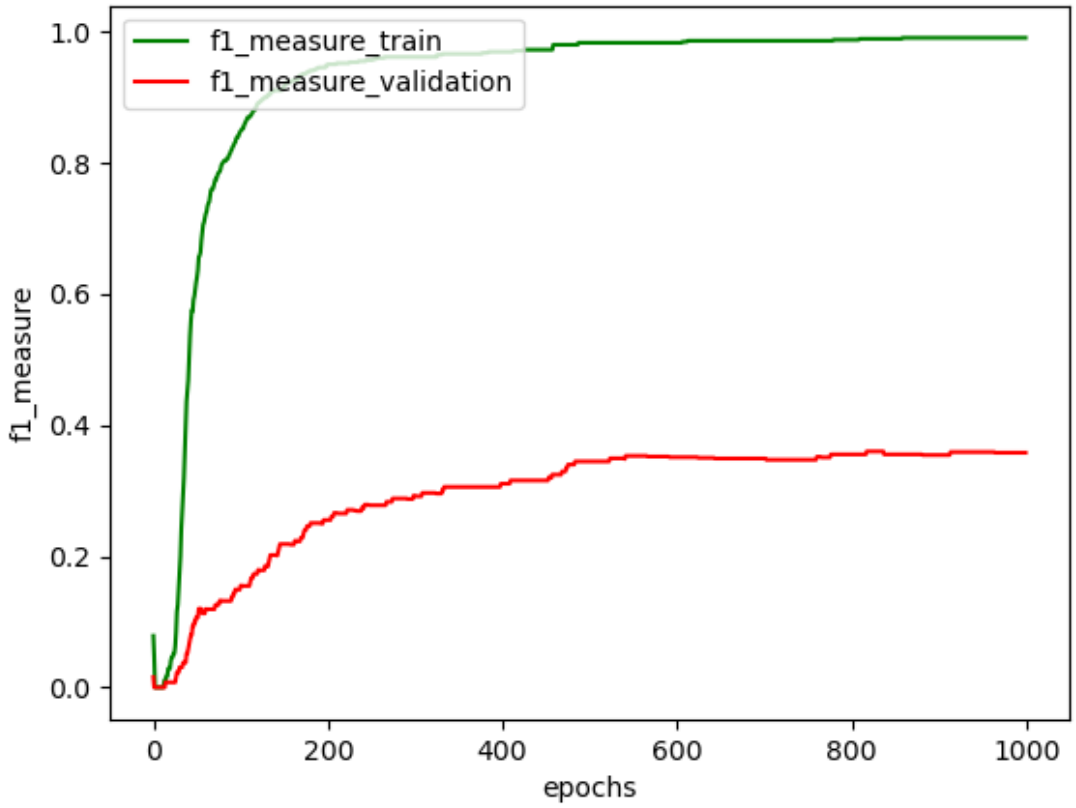
- 通过`torch.nn.BCELoss`计算目标值和预测值之间的二进制交叉熵损失函数，通过`torch.optim.Adam`优化参数
- 在训练集中，首先通过前向传播计算sigmoid激活函数，计算其交叉熵损失函数，然后计算其f1 measure，再通过后向传播更新参数。
- 在验证集中，利用刚刚更新的参数通过前向传播计算其sigmoid激活函数和交叉熵损失函数，计算其f1 measure。
- 迭代终止后，通过模型计算测试集的sigmoid激活函数，计算其f1 measure。

```
epochs:1000 loss:57.46478271484375
train:
precision_ratio=1.0 recall_ratio=0.9824046920821115 f1_measure=0.9911242603550295
validation:
precision_ratio=0.5855855855855856 recall_ratio=0.25793650793650796 f1_measure=0.35812672176308546
test:
precision_ratio=0.5945945945945946 recall_ratio=0.22 f1_measure=0.32116788321167883
```

根据结果显示，迭代1000次后，训练集的f1 measure大致收敛于0.99，验证集的f1 measure大致收敛于0.35，测试集f1 measure为0.32。

下图为利用`matplotlib`库绘制的loss曲线和f1 measure曲线。可直观看出，在梯度下降的过程中，训练集和测试集的loss函数值逐渐降低，最终趋于稳定，而**f1 measure** 逐渐升高,最终趋于稳定，达到训练目的。


loss_train
loss_validation
loss
epochs
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000


f1_measure_train
f1_measure_validation
f1_measure
epochs
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000

# 3总结反思

可以看出无论是手动求导还是自动求导，其验证集的f1 measure值都较低，训练结果不是很理想，笔者估计和原始数据有关系，训练数量太少，导致参数并没有达到最优，从而导致验证集结果不理想。