作业三实验报告

1854116 朱明志

目录

1 数据集描述

1.1 研究背景

全球患终末期肾病（ESRD）的人口每年都在增加。血液透析可提供极好的、快速的溶质清除，是需要透析者最常用的肾脏替代疗法。血压（BP）变化是经常遇到的，并与常规血液透析（HD）期间的大多数并发症有关。透析内低血压和高血压都对不良结果有强烈的影响，包括增加心血管事件和总体发病率和死亡率。

然而，透析相关的低血压被认为是一种频繁的并发症，在每个 HD 会话发生概率范围从 5.6 到 76.7%，并可能增加症状负担。一份报告显示，75%的 HD 患者至少有一个低血压的情节。另一项研究显示，超过 50%的治疗因肌内低血压而复杂化。髓内低血压与老年糖尿病患者和较低的透析前血压和较长的透析年份有关。虽然确切的机制是很难理解的，可能的机制包括增加的左心室质量指数，细胞外体积超载，交感神经过度活跃，钠负荷从透析液，和使用抗高血压药物-期间。

在初始透析过程中发生的血容量减少可导致低血压。肾脏疾病结果质量倡议(KDOQI)指南将尿内低血压定义为收缩压(SBP)下降>20mmHg 或平均动脉血压下降>10mmHg，与症状相关。然而，由于在大型数据库中通常无法获得症状和干预数据，一些尿内低血压定义完全基于 SBP 测量。患有低血压的患者在 BP 下降之前可能会出现肌肉痉挛、恶心、呕吐、打哈欠、叹气、头晕和声音嘶哑等症状。这些症状在常规透析过程中很容易被忽视。

已经提出了一些策略来预防透析间低血压的风险。这些策略包括尽量减少透析间体重增加，透析前停用降压药，透析期间不进食。然而，这些方法并没有明显降低透析间低血压的发生率。因此，避免和/或降低这种严重并发症的发生频率是透析中心的医生和护士的一个重要问题[1]。

1.2 数据集组成

本次作业采用的数据集主要来自于共 595 名慢性肾病患者的血液透析治疗数据，包括患者基本数据，血透治疗单数据，透析中记录数据，实验室检查数据。

- 患者基本数据

该数据集中包括总共 595 名患者的基本信息数据，共有数据 595 条，数据格式如下：

字段名称：病人 id,sex,birthday

示例记录：2020097,男,1978-11-25

字段说明：病人唯一对应 id,病人性别，病人生日

- 血透治疗单数据

该数据集中包括总共 595 名患者每次透析开始前的检测数据，共有数据 60619 条，数据格式如下：病人 id,examineDate,参数记录,透前体重，透析前收缩压，开始时间，体重变化

示例记录：（注：在该文件中一条记录被分拆为多行数据，每行数据只记录了一个数据点的数据，因此请依据相同的病人 id 与 examineDate 将相应的多行数据合并为一行数据后再使用）

2513,2019-01-30 13:24,,68.3,,,

2513,2019-01-30 13:24,,,,,1.8

2513,2019-01-30 13:24,,,,2019-01-30 13:00,

2513,2019-01-30 13:24,0.0,,,,

字段说明：病人唯一对应 id，透析前检查的时间，该次透析治疗的 id，透析前病人体重，透析前病人收缩压，透析开始时间，透析前后体重变化

- 透析中记录数据

该数据集中包括总共 595 名患者在每次血液透析治疗过程中的参数记录数据，共有数据 60583 条，数据格式如下：

字段名称：病人 id,记录时间,收缩压,收缩压数值,超滤率,体温,电导度,透析液温度,参数记录

示例记录：2513,2019-01-30 13:00,否,132.0,600.0,36.0,14.0,36.5,0

字段说明：病人唯一对应 id，透析中数据记录时间，当前时刻收缩压是否处于高风险的二分类变量，收缩压数值,超滤率,体温,电导度,透析液温度，该条记录对应的透析治疗的 id

2 数据清洗与数据预处理

原始数据集包含 3 个文件，每个文件都记录了一些患者的信息，现在对这三个文件进行整合和清洗以方便后续步骤更方便的进行作业任务执行。

2.1 患者基本数据文件处理

处理后的数据文件保存在 outputcsv 子文件夹下的 mid1.csv 文件中。

2.1.1 字符属性值转换

观察这个文件中的三列属性，病人 ID 是病人的主键不能随意更改，但是“男“、”女”这两个分类标签明显不易作处理，故对这两个标签使用 0,1 数值做替换。

2.1.2 生日信息转年龄年龄信息

观察另外两个文件，发现这些病人记录的透析治疗时间全部在 2019 年。对于我们来说生日信息反应的特征并不明显，而文件记录的透析治疗年份是固定的。因此可以通过 2019 减去这些病人的出生年信息，得到病人在接受透析治疗时候的年龄。

例如：

生日	透析治疗时年龄
1941/3/21	78

2.2 患者血透治疗单数据文件处理

利用 groupby 方法，将源文件中的 5 行数据合并为 1 行，最后得到一个稠密数据集文件。处理前后对比见下表：

处理前数据						
病人id	examineDate	参数记录	透前体重	透析前收缩压	开始时间	体重变化
2513	2019/1/30 13:24		68.3			
2513	2019/1/30 13:24					1.8
2513	2019/1/30 13:24				2019/1/30 13:00	
2513	2019/1/30 13:24	0				
2513	2019/1/30 13:24			132		
处理后数据						
参数记录	透前体重	透析前收缩压	体重变化	病人id		
0	68.3	132	1.8	2513		

开始时间是一个冗余数据属性，因为这一列数据可以通过参数记录这个索引在透析中记录数据文件中找到，故将其删除。其余属性将其分组整合后保留含非空行的值，处理后的数据文件有 12320 行，保存在 outputcsv 子文件夹下的 mid2.csv 文件中。

2.3 透析中记录数据文件处理

本节处理结果保存在 outputcsv 子文件夹下的 mid3.csv 文件中。

2.3.1 收缩压数据属性删除

可以看到任务中的分类标准均为讨论收缩压的范畴，而收缩压这一列属性与其他列属性无明显关联，是孤立的一列且：

```
否    59099
是     1428
Name: 收缩压,
```

可以看到这个属性列还是一个近零方差属性，所以将这个属性列删除。

2.3.2 缺失值数据行删除

将文件中的空值替换为 np.nan 型的缺失值，之后对缺失值进行统计,统计结果如下：

```
病人id       0
记录时间       1
收缩压数值     92
超滤率        18
体温          5
电导度        16
透析液温度      2
参数记录        0
dtype: int64
```

可见数据中存在极少量缺失值，将这些包含缺失值的行直接删除是最直接有效的方法。

2.4 三数据表连接

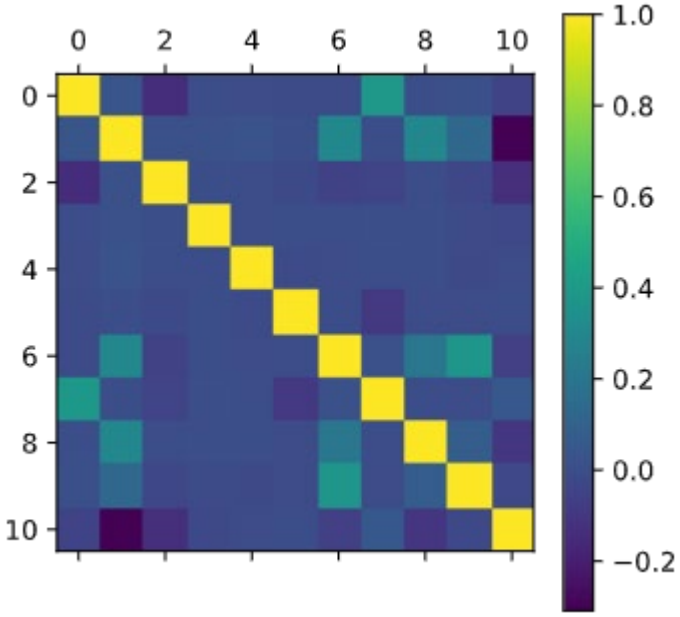
正如处理血透数据单数据表时候所述，参数记录是一个索引，一次透析活动对应一个参数记录值，通过这个值我们可以将数据表血透治疗单和数据表透析中记录连接起来，这样对于在透析过程中的每一条实时记录我们都可以有透前体重、透前收缩压、体重变化等属性可以参考了。病人 ID 列有且仅保留一列，删除无用的记录时间，因为数据集本身的排列就是按照透析数据的记录顺序排列的，时间序列信息是存在的。而具体的时间对研究收缩压变化无直接关联且这部分的分类信息还被包含在了参数记录属性中了，故对此列数据也选择删除。（但是为了最后结果文件中的记录时间这一个属性列仍然被保留在中间文件中，只是不参与相应的操作）

最后利用病人 ID 将每个病人的生日和性别也加入最终的数据表中，得到三表信息联合的一个综合信息表。这个处理之后的数据集含有 60399 条透析数据，12 种属性。

本小节的处理结果保存在 outputcsv 子文件夹下的 out.csv 文件中。

2.5 相关系数

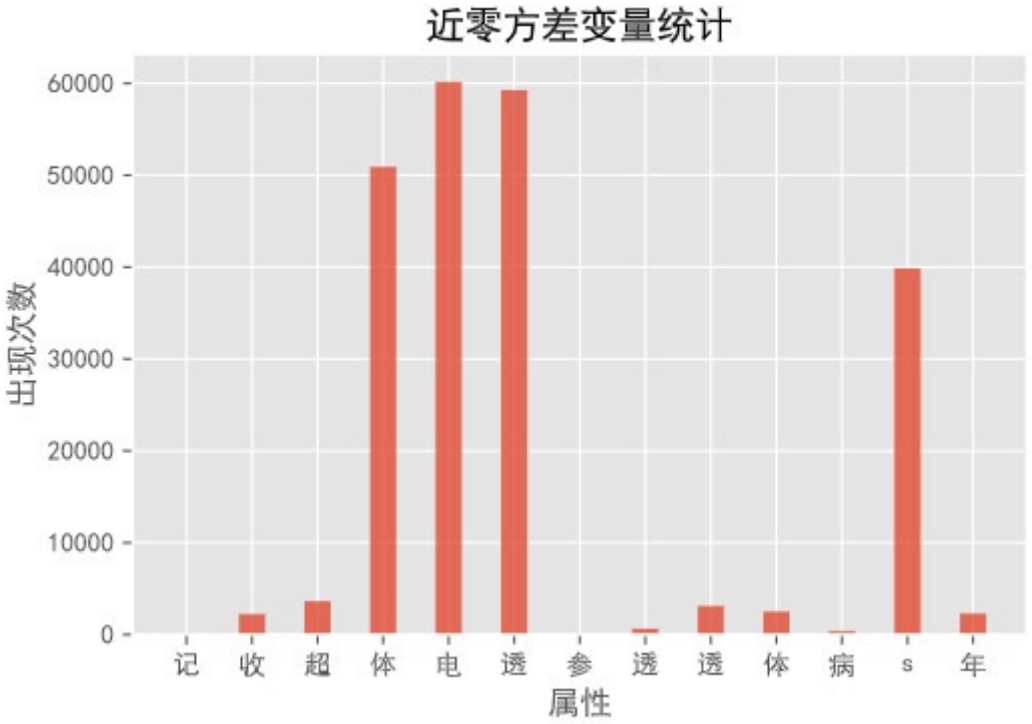
一个需要探索的数值变量是属性之间的相关性。下面的热力图以绝对值显示了这些相关性，当颜色从深紫色变化到黄色时，相关性的绝对值从 0 增长到 1。下图统计了两两属性之间的相关系数大于 0.8 的组数为 0 组：一般来说，这些属性没有很强的相关性。



3 任务 1：特征选取

3.1 近零方差列分析

对 out 数据表进行近零方差列分析，使用 pandas 的 value_count 方法，统计每一个属性中每个属性值出现的次数，并将出现次数最多属性值的次数记录下来，用一个直方图展现出来：



可见电导度和透析液温度这两个属性中某一个属性值都代表了绝大多数患者接受治疗时候的情况。比如电导度这个属性，总共只有 60399 个数据，有 60157 个患者设置的值都为 14.0，剩下的情况不足总数 1%，我甚至可以认为其余值是统计学上的错误数据，同理透析液温度也是如此有 59261 个记录都设置了 36.5 这个数值，剩下的情况不足总数的 2%。故这两个属性是近邻方差属性，加入后面的分类依据毫无意义，甚至可能造成过拟合，在特征选取的时候舍弃考虑这列属性。

3.2 标记属性删除

这一部分探索的是删除那些用于区分病人和不同次透析标签的属性列，具体表现在病人 id 和记录时间两个属性列上。这两个属性是标记属性对分类结果无实际意义上的贡献，因此将其删除避免模型过拟合。

3.3 收缩压数据整合

研究每一列数据属性代表的含义，可以发现收缩压数值列与透前收缩压之差即为当前患者在本次透析治疗过程中收缩压的变化值，以此可以作为判断依据替代当前收缩压属性，同时删除透前收缩压属性。

3.4 最终特征选择结果

最终从数据表中选取了 7 种无强相关性，且有实际医学意义的属性作为分类特征，它们是：

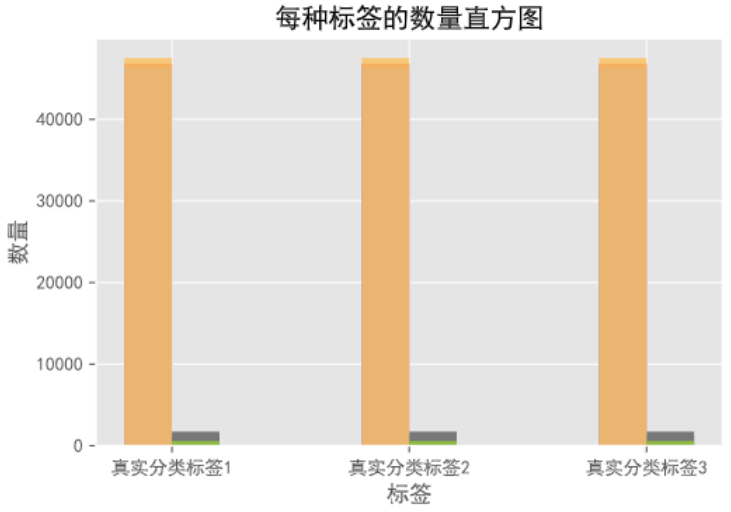
收缩压变化数值、超滤率、体温、透前体重、体重变化、sex、年龄

4 任务 2：三种不同标准的分类标签生成

这一部分的具体实现见 Python 原码。三种标签通过三个自定义的函数生成，生成规则严格依照任务要求。评为低风险的数据行标签热编码为数字 0，评为高风险的数据行标签热编码为数字 1。这部分的成果被用于下一步训练分类器，具体标签保存在主文件夹下的 result.csv 文件中，这些标签数量的分布如下：

（特别说明的每次透析治疗的最后一次记录数据因为不符合数据标签生成规则，这一行数据被删除了，删除后的数据有 48098 条）

```
0.0    46814
1.0     1284
Name: 真实分类标签1, dtype: int64
0.0    46366
1.0     1732
Name: 真实分类标签2, dtype: int64
0.0    47518
1.0      580
Name: 真实分类标签3, dtype: int64
```



5 任务 3：选取三种分类器，分类数据

依据任务 2 中生成的三种标签，每种标签训练一个分类器并以此分类数据。本节所述的所有神经网络都基于 Keras 框架搭建，运行的环境背景是 TensorFlow2.4.1，支持的 CUDA 版本是 11.2。

5.1 LOGISTIC REGRESSION

5.1.1 分类器搭建

逻辑回归虽然被称为回归，但其实际上是分类模型，并常用于二分类。逻辑回归因其简单、可并行化、可解释强深受工业界喜爱。逻辑回归的本质是：假设数据服从这个分布，然后使用极大似然估计做参数的估计。

以二分类为例，对于所给数据集假设存在这样的一条直线可以将数据完成线性可分。决策边界可以表示为$w_1x_1 + w_2x_2 + \mathrm{b} = 0$，假设某个样本点有$h_w(x) = w_1x_1 + w_2x_2 + \mathrm{b} > 0$，那么可以判断它的类别为 1，这个过程其实是感知机。

逻辑回归还需要加一层，它要找到分类概率 P(Y=1) 与输入向量 x 的直接关系，然后通过比较概率值来判断类别。

考虑二分类问题，给定数据集：

$$\mathrm{D} = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N), x_i \subseteq R^n, y_i \in 0,1, \mathrm{i} = 1,2, \cdots, \mathrm{N}$$

考虑到$w^T\mathrm{x} + \mathrm{b}$取值是连续的，因此它不能拟合离散变量。可以考虑用它来拟合条件概率$\mathrm{p}(Y = 1 \mid x)$，因为概率的取值也是连续的。

但是对于$\mathrm{w} \neq 0$（若等于零向量则没有什么求解的价值），$w^T\mathrm{x} + \mathrm{b}$取值为 R，不符合概率取值为 0 到 1，因此考虑采用广义线性模型。

最理想的是单位阶跃函数：

$$p(y = 1 \mid x) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0, z = w^Tx + b \\ 1, & z > 0 \end{cases}$$

但是这个阶跃函数不可微，对数几率函数是一个常用的替代函数：

$$y = \frac{1}{1 + e^{-(w^Tx + b)}}$$

综上理论所述，我可以搭建这样一个 3 层的神经网络来实现逻辑回归分类器：

神经网络的输入层有 7 个输入神经元对应 7 种不同属性，有一个隐藏层，该隐藏层的激活函数选用 sigmoid 函数，即上文提到的替代函数，输出层设置 2 个神经元，激活函数选用 softmax 函数，标签在输入前进行 2 进制标签拆分（0 标签对应 10,1 标签对应 01），输出预测标签进行 2 进制整合（10 输出对应 0,01 输出对应 1）。

对整个数据集进行 82 拆分，选取 20%的数据作为验证集，选取 80%的数据作为训练集，最后将训练好的分类器对整个数据集进行分类标签生成，这一部分的预测分类标签保存在 result.csv 文件中。

神经网络结构如下：

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 28)                224
_________________________________________________________________
activation (Activation)      (None, 28)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 58
_________________________________________________________________
activation_1 (Activation)    (None, 2)                 0
=================================================================
Total params: 282
Trainable params: 282
Non-trainable params: 0
```

这个模型被保存 models 子文件下，包括了训练后的参数，为一个 h5 格式的模型文件。

5.1.2 分类器参数设置

我们是一个二分类问题，很明显，binary crossentropy 是一个好的损失函数选择。即：

$$J(w) = -\frac{1}{n}\left(\sum_{i=1}^{n}\left(y_i \ln\, p(x_i) + (1-y_i)\ln\left(1-p(x_i)\right)\right)\right)$$

对于正则器的选择，我使用 Keras 框架下自带的 L1，L2 正则规则，正则系数均设置为 0.5。

每轮训练结束后我设置了 shuffle 参数，自动打乱数据集，保证训练效果。

同时梯度下降窗口数值设置为 9 个数据一次，这是遍历 1-99 后的最佳参数。

一共训练 4 轮，主优化器选择 RMSprop 优化器，学习率为 0.0001。

特别的对于这个不均衡的分类标签，为了保证分类效果，我们采用了标签权重设置，保证不会因为数据量的差异导致神经网络学习效果太差。对于 class_weight 类权重设置，由自己编写的一个分类指标平衡函数生成类的权值，这样可以起到平衡数据集的作用。（低血压和高血压数据量差异过大，见第 4 节）平衡标签生成函数实现见下，利用 sklearn 包类的 class_weight 方法实现：

```
def bal(y):
    class_weight1 = class_weight.compute_class_weight('balanced',
                                                      np.unique(y),
                                                      y)
```

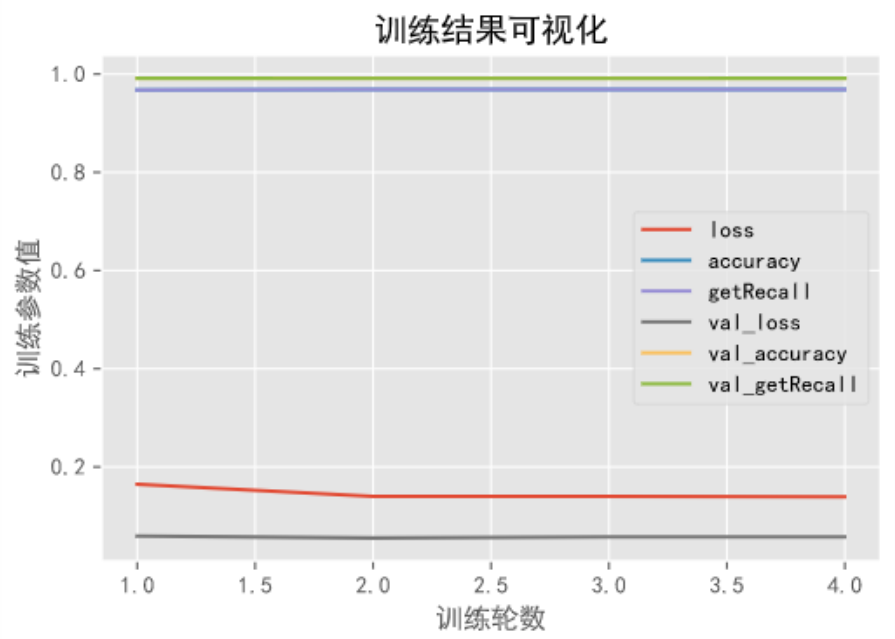
```
        cw = dict(enumerate(class_weight1))
        return cw
```

5.1.3 分类器训练过程可视化

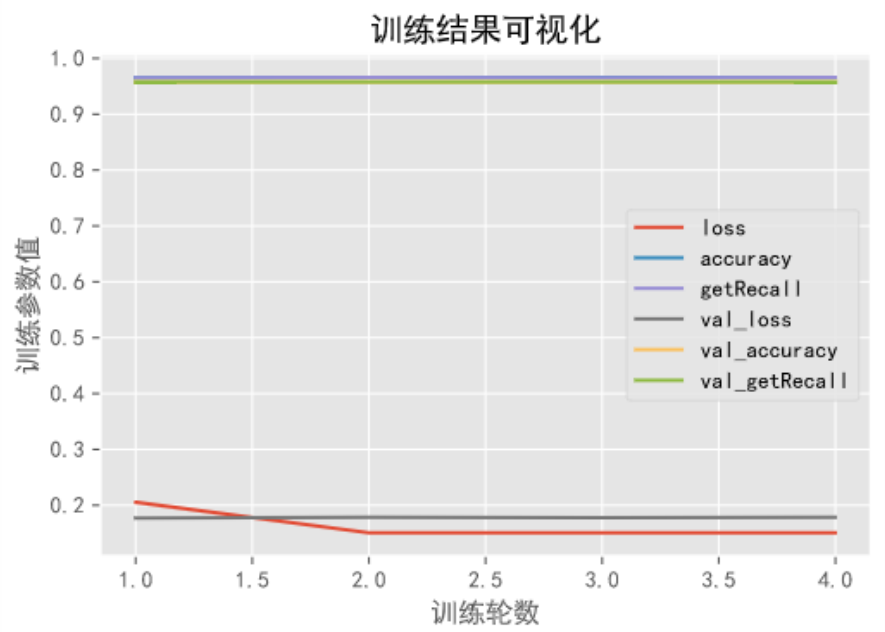
逻辑回归分类器经过 4 轮训练，每轮训练结束后评估训练的结果，评估结果如下：

评估手段	含义
Loss	损失函数
Accracy	准确率
getRecall	Recall 值
Val_*	测试集上相应参数

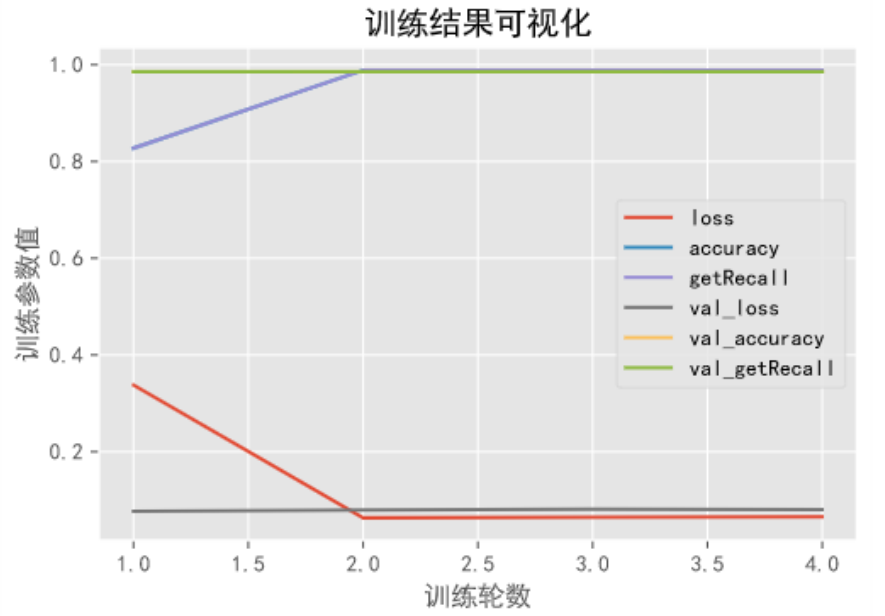
- 分类标签 1



- 分类标签 2



- 分类标签 3



5.2 SVM

5.2.1 分类器搭建

使用神经网络可以模拟 SVM 分类器,具体操作是使用一个自定义的损失函数替换原有的 2 种交叉熵损失函数，并且自规定一个正则方法，以此搭建一个单隐藏层的神经网络。选择$f(x) = x$形式的函数作为隐藏层的激活函数，输出层的神经元数量依然取决于实际分类需求的类别数量。主要使用的是 SVM 的 hinge loss 形式的损失函数作为本自搭建神经网络的损失函数。

原始的 SVM 的损失函数[2]：

$$\min_{\mathbf{w},\xi_n} \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t.} \quad \mathbf{w}^\top\mathbf{x}_n t_n \geq 1 - \xi_n \forall n$$

$$\xi_n \geq 0 \forall n$$

神经网络模拟 SVM 时的类 hinge loss[2]：

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{n=1}^{N} max(1 - \mathbf{w}^\top\mathbf{x}_n t_n, 0)$$

观察两式可知，它其实可以看做是一个 hinge loss 加上一个 L2 正则方法，前面的 1/2 就是 L2 正则项的系数 lambda，这在神经网络中很容易实现。

在 keras 中添加一个自定义损失函数，利用这个损失函数训练模型，这个自定义损失函数实现如下：

```
def categorical_squared_hinge(y_true, y_pred):
    y_true = 2. * y_true - 1
    # trans [0,1] to [-1,1]，注意这个，svm 类别标签是-1 和 1
    # hinge loss，参考 keras 自带的 hinge loss
    vvvv = K.maximum(1. - y_true * y_pred, 0.)
    vvv = K.square(vvvv)
    # 文章《Deep Learning using Linear Support Vector Machines》有进行平
方
    vv = K.sum(vvv, 1, keepdims=False)
    # axis=len(y_true.get_shape()) - 1
    v = K.mean(vv, axis=-1)
    return v
```

综上理论所述，我可以搭建这样一个 3 层的神经网络来实现 SVM 分类器：

神经网络的输入层有 7 个输入神经元对应 7 种不同属性，有 1 个隐藏层，该隐藏层的激活函数选用 linear 函数，即上文提到的原值函数，输出层设置 2 个神经元，激活函数选用 softmax 函数，标签在输入前进行 2 进制标签拆分（0 标签对应 10,1 标签对应 01），输出预测标签进行 2 进制整合（10 输出对应 0,01 输出对应 1）。

对整个数据集进行 82 拆分，选取 20%的数据作为验证集，选取 80%的数据作为训练集，最后将训练好的分类器对整个数据集进行分类标签生成，这一部分的预测分类标签保存在 result.csv 文件中。

神经网络结构如下：

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 7)                 56
_________________________________________________________________
activation_12 (Activation)   (None, 7)                 0
_________________________________________________________________
dense_13 (Dense)             (None, 2)                 16
_________________________________________________________________
activation_13 (Activation)   (None, 2)                 0
=================================================================
Total params: 72
Trainable params: 72
Non-trainable params: 0
```

这个模型被保存 models 子文件下，包括了训练后的参数，为一个 h5 格式的模型文件。

5.2.2 分类器参数设置

损失函数是我们自定义的一个模拟 SVM 分类器逻辑的损失函数，即：

$$J(w) = C\sum_{n=1}^{N} max(1 - \mathbf{w}^{\top}\mathbf{x}_n t_n, 0)$$

对于正则器的选择，我使用 Keras 框架下自带的 L2 正则规则，正则系数均设置为 0.5，以模拟 SVM 分类器的正则规则。正则规则和损失函数结合起来模拟了 SVM 分类的逻辑。

每轮训练结束后我设置了 shuffle 参数，自动打乱数据集，保证训练效果。

同时梯度下降窗口数值设置为 50 个数据一次，这是遍历 1-99 后的最佳参数。

一共训练 8 轮，主优化器选择 RMSprop 优化器，学习率为 0.0001。

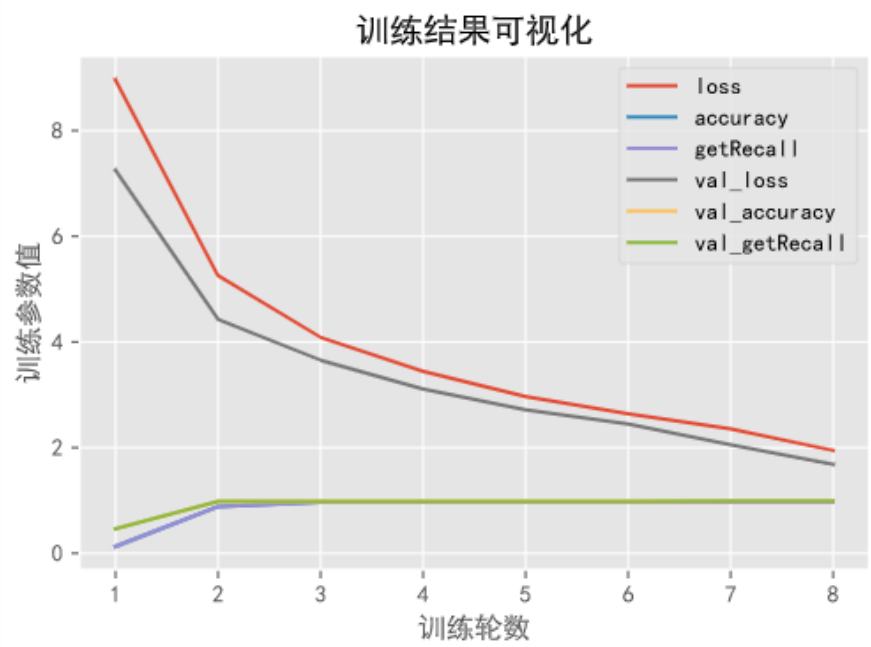
特别的对于这个不均衡的分类标签，为了保证分类效果，我们采用了标签权重设置，保证不会因为数据量的差异导致神经网络学习效果太差。对于 class_weight 类权重设置，由自己编写的一个分类指标平衡函数生成类的权值，这样可以起到平衡数据集的作用。实现见 5.1.2。

5.2.3 分类器训练过程可视化

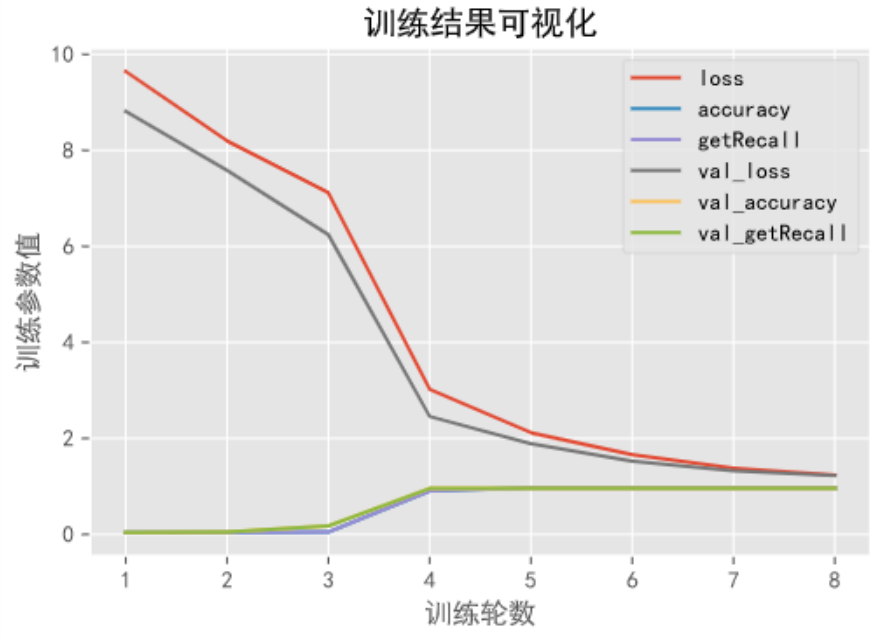
逻辑回归分类器经过 8 轮训练，每轮训练结束后评估训练的结果，评估结果如下：

评估手段	含义
Loss	损失函数
Accracy	准确率
getRecall	Recall 值
Val_*	测试集上相应参数

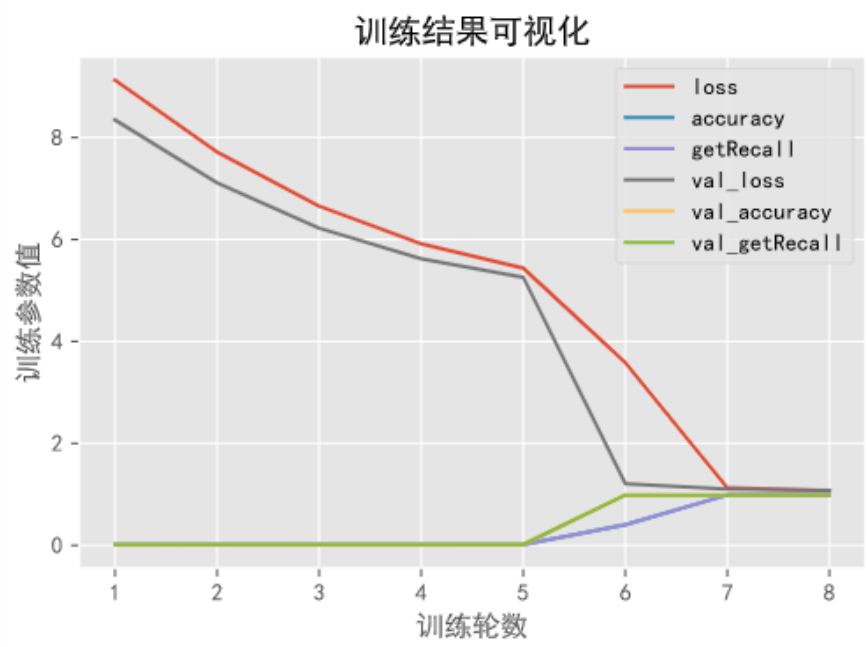
- 分类标签 1


训练结果可视化
loss
accuracy
getRecall
val_loss
val_accuracy
val_getRecall
训练参数值
训练轮数
0
2
4
6
8
1
2
3
4
5
6
7
8

- 分类标签 2


训练结果可视化
loss
accuracy
getRecall
val_loss
val_accuracy
val_getRecall
训练参数值
训练轮数
0
2
4
6
8
10
1
2
3
4
5
6
7
8

- 分类标签 3



5.3 ANN 4 层神经网络

5.3.1 分类器搭建

基于前两个分类器的搭建经验，我可以搭建这样一个 6 层的神经网络来实现简单 ANN 分类器：

神经网络的输入层有 7 个输入神经元对应 7 种不同属性，有 4 个隐藏层，该隐藏层的激活函数选用 LeakyReLU 函数，输出层设置 2 个神经元，激活函数选用 softmax 函数，标签在输入前进行 2 进制标签拆分（0 标签对应 10,1 标签对应 01），输出预测标签进行 2 进制整合（10 输出对应 0,01 输出对应 1）。

对整个数据集进行 82 拆分，选取 20%的数据作为验证集，选取 80%的数据作为训练集，最后将训练好的分类器对整个数据集进行分类标签生成，这一部分的预测分类标签保存在 result.csv 文件中。

神经网络结构如下：

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 144)               1152
_________________________________________________________________
dense_1 (Dense)              (None, 72)                10440
_________________________________________________________________
dense_2 (Dense)              (None, 36)                2628
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 74
_________________________________________________________________
activation (Activation)      (None, 2)                 0
=================================================================
Total params: 14,294
Trainable params: 14,294
Non-trainable params: 0
```

5.3.2 分类器参数设置

我们是一个二分类问题，很明显，binary crossentropy 是一个好的损失函数选择。即：

$$J(w) = -\frac{1}{n}\left(\sum_{i=1}^{n}\left(y_i \ln\ p(x_i) + (1-y_i)\ln\left(1-p(x_i)\right)\right)\right.$$

对于正则器的选择，我使用 Keras 框架下自带的 L1，L2 正则规则，正则系数均设置为 0.5。

每轮训练结束后我设置了 shuffle 参数，自动打乱数据集，保证训练效果。

同时梯度下降窗口数值设置为 20 个数据一次，这是遍历 1-99 后的最佳参数。

一共训练 6 轮，主优化器选择 Adam 优化器。

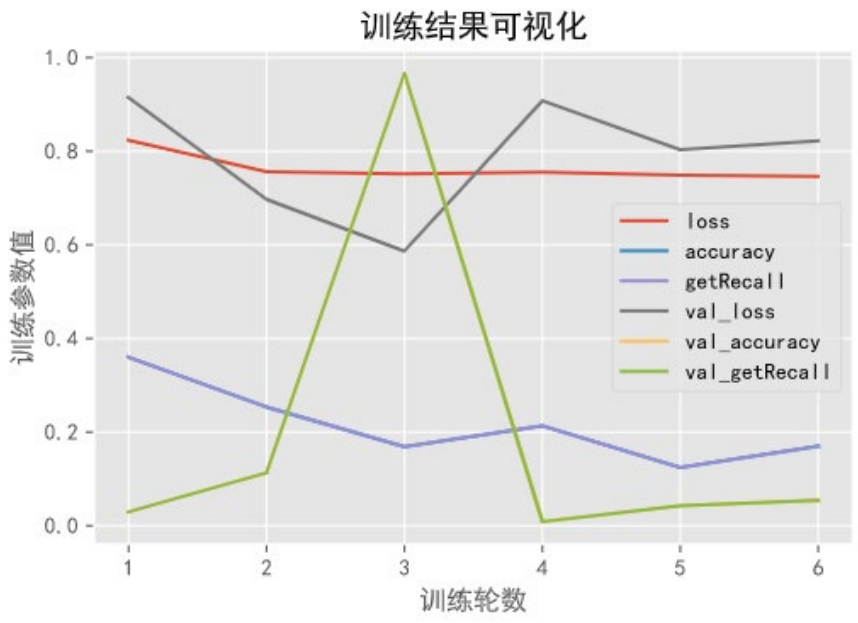
特别的对于这个不均衡的分类标签，为了保证分类效果，我们采用了标签权重设置，保证不会因为数据量的差异导致神经网络学习效果太差。对于 class_weight 类权重设置，由自己编写的一个分类指标平衡函数生成类的权值，这样可以起到平衡数据集的作用。实现见 5.1.2。

5.3.3 分类器训练过程可视化

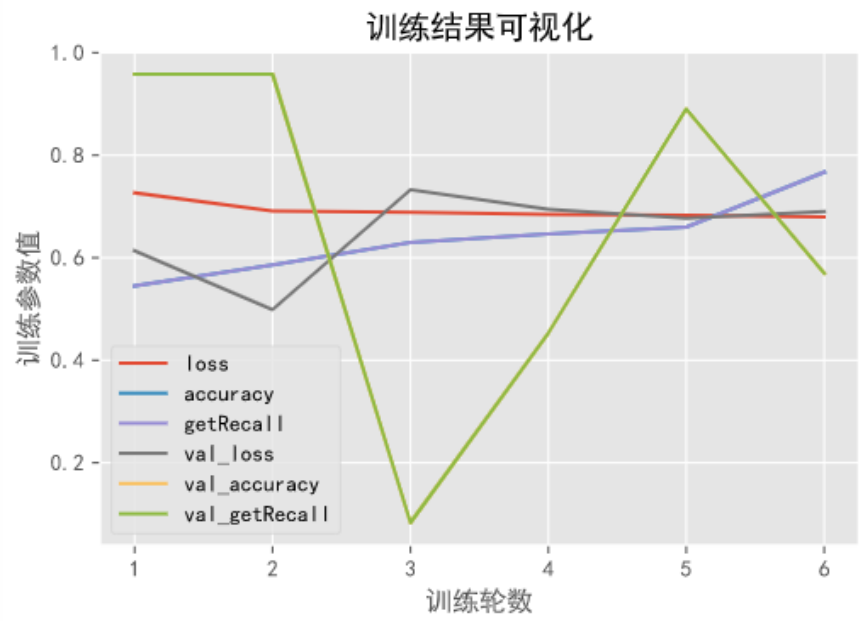
逻辑回归分类器经过 6 轮训练，每轮训练结束后评估训练的结果，评估结果如下：

评估手段	含义
Loss	损失函数
Accracy	准确率
getRecall	Recall 值
Val_*	测试集上相应参数

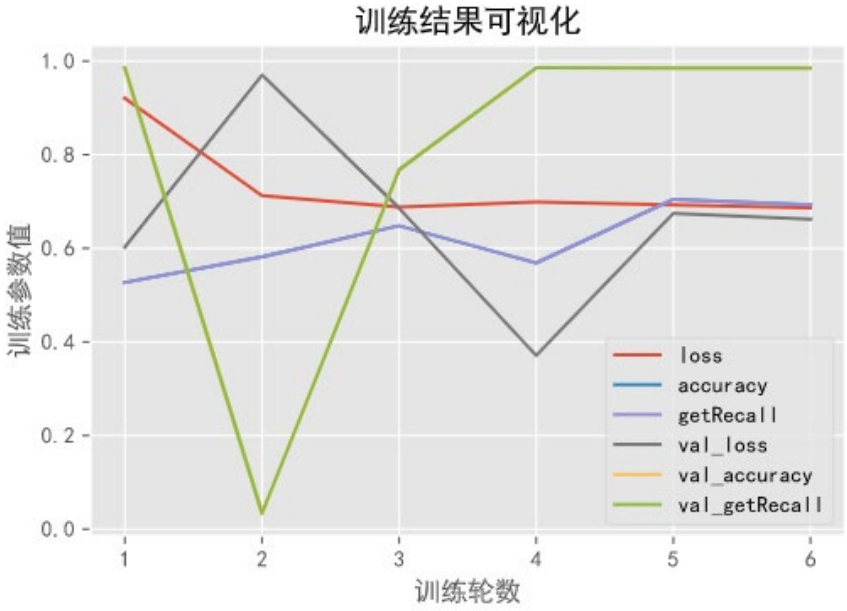
- 分类标签 1


训练结果可视化
1.0
0.8
0.6
0.4
0.2
0.0
训练参数值
loss
accuracy
getRecall
val_loss
val_accuracy
val_getRecall
1
2
3
4
5
6
训练轮数

- 分类标签 2


训练结果可视化
1.0
0.8
0.6
0.4
0.2
训练参数值
loss
accuracy
getRecall
val_loss
val_accuracy
val_getRecall
1
2
3
4
5
6
训练轮数

- 分类标签 3



6 实验结果分析与讨论

这次作业使用 keras 手动搭建了三种神经网络，其中两种模拟传统分类器，取得了一定的成果。通过这次实验，我对如何调节 Keras 框架下各模型参数有了更深入的理解，自己实现了一个 get recall 评价函数和一个损失函数，对逻辑回归和 SVM 分类器的原理有了更深入的理解。

6.1 各模型参数总览

模型名称	损失函数选择	正则器	训练轮数	梯度下降窗口大小	优化器
LR 逻辑回归	Binary-crossentropy	L1，L2	4	9	RMSprop
SVM	SVM 类损失函数	L2	8	50	RMSprop
ANN	Binary-crossentropy	L1，L2	6	20	Adam

6.2 最终分类结果

6.2.1 分类 1

分类算法	Loss Function	Accuracy	Recall
LR 逻辑回归	0.7374	0.7202	0.7203
SVM	1.1784	0.9685	0.9685
ANN	0.7233	0.1613	0.1613

6.2.2 分类 2

分类算法	Loss Function	Accuracy	Recall
LR 逻辑回归	0.7074	0.9640	0.9640
SVM	1.2991	0.9650	0.9650
ANN	0.6586	0.5467	0.5467

6.2.3 分类 3

分类算法	Loss Function	Accuracy	Recall
LR 逻辑回归	0.7278	0.9885	0.9885
SVM	1.1408	0.9882	0.9882
ANN	0.7042	0.7897	0.7897

6.3 结果分析

SVM 分类器的训练结果最标准，训练下降曲线也很标准，可见有较好的表现。实际中也确实有较好的准确率。自己搭建的 4 层神经网络在分类标签 1 上有着几乎失败的表现，可能是这个网络结构不太适合这个数据，这个网络结构是我随意搭建的，可能需要找一个现成的适合这个数据集的网络结构来作为分类器。后续研究可以手动搭建一个更为复杂更深的网络结构来进行实验，这次作业受限于时间原因仅是实现了这样一个简单的 ANN 模型。

7 参考文献

[1] Lin C J， Chen Y Y， Pan C F , et al. Dataset supporting blood pressure prediction for the management of chronic hemodialysis[J]. Scientific Data, 2019, 6(1):313.

[2] Tang Y . Deep Learning using Linear Support Vector Machines[J]. Computer ence, 2013.