# 关于学习统计学习方法的一些思考与总结

2021 年 2 月 2 日

## 1　统计学习三要素

所谓的统计学习方法，指的是一系列用于从数据中挖掘出有效信息和知识的方法，每一种统计学习方法都可看成是由三个要素组成的：１）模型；２）策略；３）算法．模型就是一个带有参数的函数

$$f_{\boldsymbol{\omega}} : \mathcal{I} \longrightarrow \mathcal{O} \tag{1.1}$$

式中：$\mathcal{I}$ 是它的输入范围，$\mathcal{O}$ 是输出范围，$\boldsymbol{\omega}$ 是参数，对于不同的参数 $\boldsymbol{\omega}$，函数 $f_{\boldsymbol{\omega}}$ 在面对即使是同样的输入时，输出也可能有所不同，换句话说，$\boldsymbol{\omega}$ 有可能会影响 $f_{\boldsymbol{\omega}}(\boldsymbol{x})$，这里 $\boldsymbol{x} \in \mathcal{I}$．既然由不同的 $\boldsymbol{\omega}$ 就可以得到不同的函数 $f_{\omega}$，那么我们可以说，所有 $\boldsymbol{w}$ 可能取值的取值范围 $\mathcal{P}$ 张成了一个决策空间

$$\{f_{\boldsymbol{\omega}} : \boldsymbol{\omega} \in \mathcal{P}\} \tag{1.2}$$

这里 $\mathcal{P}$ 又称为参数空间．这就引出了「策略」，它是用来评判一个决策空间中的函数 $f_{\boldsymbol{\omega}}$ 好或者不好的评判准则，举例来说，可以想象两个函数 $f_1, f_2$，它们的输入是房屋的一些信息，比如说面积，地段，修建时间，建造成本等等，而输出都是房屋的售价，假设它们各自的参数都确定了（从而模型也就确定了），那么策略是什么呢？我们希望这个房屋估价函数估得越「准确」越好，所以说估价估得是否「准确」，就能够用来评判模型的好与坏，所以说「准确度」就可以作为一个「策略」，具体是准确度的一个量化指标，比如说均方误差

$$\text{MeanSquaredErrors}(\boldsymbol{\omega}) = \frac{1}{n} \sum_{\boldsymbol{x} \in \mathcal{I}} \left(f_{\boldsymbol{\omega}}(\boldsymbol{x}) - t(\boldsymbol{x})\right)^2 \tag{1.3}$$

这个我们后面会讲到．当模型和算法都确定了，还差什么呢？我们知道什么样的模型是好的，还要知道具体该怎么样去找到这个模型，换句话说，就是一步一步把理想的那个 $\boldsymbol{\omega}$「算出来」或者找出来的步骤与方法，这样的东西就是三要素中的「算法」，它和计算机课程中的算法是一回事，只不过其中有一些算法普遍地被人们用来寻找最优的模型参数，因为它们擅长这样做．

## 2　一个案例

有一个未知的函数 $t$，以及在它身上做了 $N$ 次试验得到的实验结果 $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$，这个未知函数 $t$ 可以看做是一个未知系统的某一种性质或者属性，我们打算通过研究这个函数 $t$ 来对这个未知的系统进行研究．我们的研究方法主要是一些统计学习方法，具体地，让我们来考虑模型１：

$$f_1(x) = \omega_0 + \omega_1 x_1 \tag{2.1}$$

这就是一个简单的模型，并且模型１就可以用来当做三要素中的模型，策略是什么呢？我们希望寻找这样的一组参数 $(\omega_0, \omega_1)$ 使得

$$\text{MeanSquaredErrors}(\omega_0, \omega_1) = \frac{1}{N} \sum_{i=1}^{N} \left(f_1(x_i) - y_i\right)^2 \tag{2.2}$$

尽可能地小, 所以策略就是均方误差最小化. 知道了要做什么, 怎么样去做呢? 一种方法叫做「最小二乘法」, 他就是专门用来估计这类「回归问题」下被用到的函数的未知参数的，令 $\overline{x}=\frac{1}{n}\sum_{i=1}^{N}x_i$，$\overline{y}=\frac{1}{n}\sum_{i=1}^{N}y_i$，则

$$\begin{cases}\omega_1=\dfrac{\displaystyle\sum_{i=1}^{N}(x_i-\overline{x})(y_i-\overline{y})}{\displaystyle\sum_{i=1}^{N}x_i-\overline{x}}\\ \omega_0=\overline{y}-\omega_1\overline{x}\end{cases}\tag{2.3}$$

这就是用来寻找最优参数的计算方法，简称算法.

# 3 正则化

前面的案例提到的策略都只是在已知的样本上取得尽可能小的误差，但事实上，如果将 $f_1$ 视作更一般的多项式函数

$$f(x)=\sum_{k=0}^{m}\omega_k x^k\tag{3.1}$$

随着次数 $m$ 的增高，最终可以找到一组参数 $\boldsymbol{\omega}=(\omega_0,\cdots,\omega_m)$ 使得函数曲线经过全部样本点 $\{(x_i,y_i)\}$, 也就是说, 此时均方误差为 0. 然而, 就实际而言, 经过全部已知样本点的曲线其实是很多的, 可以有千千万万条, 所以, 当我们求出了这样一组完美的参数, 很有可能真的就是求错了, 从另一个角度来说, 我们仅仅只是取到了未知系统 $t$ 的有限个数据点，如果就这么将从这已有的有限个数据点建出的模型 $(f,\boldsymbol{\omega})$，对于未知系统的更加完整的研究而言，未免言之过早或者以偏概全了，换句话说也就是，$f$ 虽然已经完全拟合了已有的全部数据，但是由于 $f$ 就这么将这有限个数据点体现出了信息作为未知系统的全部信息，我们很担心它在未知样本点上的表现也就是它未来的表现，仅为这个未知系统的全部信息，不太可能就只在这有限多个样本点上得到全部体现，那么 $f$ 学到的，就只有不完整的信息，以及一些噪音.

在三要素中，策略的具体体现就是一个损失函数，损失函数有很多，前面提到的均方误差算是一个，正则化具体就是，在损失函数中加入一个与模型复杂度正相关的量，以此来约束模型被训练得过度复杂，比如说

$$\mathrm{SSEN}=\sum_{i=1}^{N}\left(f(x_i)-y_i\right)^2+\sum_{i=0}^{m}\omega_i^2\tag{3.2}$$

就是一个加上了正则化项的残差平方和函数，如果用它作为策略，那么相比不加正则化项的策略而言，最终找到的参数的平方和会更小一些，所以模型也就更加简单.

## 3.1 L1 正则化

也称 Lasso 正则化，就是在损失函数中加入模型的所有可学习参数的绝对值之和:

$$J(\boldsymbol{\omega})=\frac{1}{2N}\sum_{i=1}^{N}\left(f(x_i)-y_i\right)^2+\lambda\sum_{j=1}^{m}|\omega|\tag{3.3}$$

式中 $\lambda$ 是人为设定的一个参数，$\lambda$ 越大，正则化的程度就「越狠」，得出的模型的复杂度就越低. 当输入特征非常多时，比如说一份数据集有成百上千个自变量，那么 L1 正则化也会变得有用，这是因为，由于所有的可学习参数都在学习策略（损失函数）中有体现，所以那些不重要的系数会被优化得尽可能地小以实现损失函数的最小化. L2 正则化就是式 3.2了，只不过要在参数平方和前面再乘上一个系数 $\lambda$.

# 4　分类任务的学习策略

## 4.1　混淆矩阵与 ROC 曲线

统计学习方法除了有三要素以外，也大体上可以分为三类：有监督学习，无监督学习和半监督学习，我们本文主要讲有监督学习，前面我们提到的房价预测函数的学习，主要是属于有监督学习，并且是有监督学习下的回归任务，现在我们来讲的是有监督学习下的分类任务.

所谓分类非常好理解：当顾客走进咖啡店的柜台，或者饭店，服务人员面对顾客一般会用「先生」或者「女士」作为称谓，那么根据外貌、衣着打扮和口音辨别性别，就是一个分类任务；当人们看到宠物，能够说出它是猫还是狗还是兔子还是其他宠物，这也是一个分类任务；计算机程序自动判断一封邮件是否是垃圾邮件，这是一个更加常见的分类任务. 分类任务的性能其实也就是分类结果对于真实结果的准确性，我们当然希望学得的模型的分类表现越准确越好，所以自然而然地，我们会希望挑选到一组参数，这组参数能使分类器准确地分类，所以说学习策略就是「准确度」最大化，这里加了引号，我们指的不是准确度 (accuracy)，而是一整类和「准确度」相关的度量，准确地说，是「准确的程度」，准确度也是「准确的程度」的一种具体度量（或者说计算方式），在这么多地「准确的程度」的度量中，并不是所有的都适合用来作为学习策略，譬如说准确度，对样本做完全部分类之后，拿模型预测结果和样本标签做比对，得到一个叫做混淆矩阵:

$$\begin{bmatrix} \text{TruePositive} & \text{FalsePositive} \\ \text{FalseNegative} & \text{TrueNegative} \end{bmatrix} \tag{4.1}$$

准确度 (accuracy) 是这样计算的

$$\text{Accuracy} = \frac{\text{TruePositive} + \text{TrueNegative}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{4.2}$$

和它词义上容易混淆的是精确率 (precision)

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \tag{4.3}$$

矩阵4.1中，TP 是模型预测为正类（阳性）且事实上为正类（阳性）的样本数，FP 是模型预测为正类（阳性）但事实上不是正类（阳性）的样本数，FN 是模型预测为负类（阴性）但实际上为正类（阳性）的样本数，TN 是模型预测为负类（阴性）并且实际上也是负类（阴性）的样本数. 咽拭子检测也算是一个分类任务，它的输出有两种，阳性——检测结果主张被检测者得了新冠，阴性——检测结果主张被检测者没得新冠，TP 就是检测结果是阳性并且实际上也是阳性，TN 就是检测结果是阴性并且实际上也是阴性，FN 就是检测结果是阴性但是实际上却是阳性！（可怕了），这属于漏检，而 FP 则是检测结果是阳性但实际上却是阴性（从而冤枉了被检测者）. 精确率也好，准确度也好，不适合作为学习策略的原因是梯度不好计算，从而不容易应用梯度下降算法（三要素中的算法）对参数进行优化.

以检测垃圾邮件 (spam) 的计算机程序为例，对于一封邮件作为输入，它的输出一般是一个概率，由程序开发人员设定一个阈值 (criterion)，这个阈值像概率一样，也是一个介于 0 到 1 的数，邮件检测程序首先计算出一封邮件是垃圾邮件的概率，然后在和阈值做比对，比如说如果小于阈值就判定不是垃圾邮件，如果大于阈值就判定是垃圾邮件，也跟咽拭子检测类似，也有判断正确的情况和误判的情况，由于真实情况只有两种——是垃圾邮件或者不是，所以当确定一个阈值后，也可以类似地计算出形如矩阵4.1那样的混淆矩阵 (confusion matrix). 事实上，这个阈值跟模型的表现也有微妙的关系，并且，它实际上还会考虑到用户的习惯：如果阈值设置得接近 1，那会发生什么？那么只有很少的邮件被判断为垃圾邮件，所以基本上不容易把一封正常邮件错怪成垃圾邮件，但是却可能把很多垃圾邮件判断为正常邮件；而如果阈值设置得非常接近 0，那么这个程序就会非常严格，可能就把很多不是垃圾邮件的也判定成了垃圾邮件，但是漏掉的垃圾邮件也就是把垃圾邮件判定为正常邮件的个案数却减少了；说了这么多，一方面是希望说明，阈

值的设置有时候会联系实际，比如说要看用户觉得把正常邮件判定为垃圾邮件的损失大，还是把垃圾邮件判定为正常邮件的损失大；另一方面，它也引出了两个概念：一个叫做精确率 (precision):

$$\text{Precison} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.4}$$

还有一个叫做召回率 (recall):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.5}$$

不难理解，如果阈值非常接近 1，那么精确率一定很高，毕竟那么高的概率怎么可能不是？但是召回率却会下降，也就是说会漏掉许多，而如果阈值非常接近 0，那么召回率就会变得很高，甚至如果阈值就设置为 0，那么召回率就直接是 100%，可是精确率又会降低了. 所以精确率、召回率其实就是前面那段话的量化表述. 那么怎么样来描述诸如「精确率和召回率都高」的程度？我们有

$$\frac{2}{\text{F-Score}} = \frac{1}{\text{Precision}} + \frac{1}{\text{Recall}} \tag{4.6}$$

F-得分 (F-Score) 被定义为精确率与召回率的调和平均，观察该式可以看到如果 Precision 和 Recall 都接近 1，那么 F-Score 也接近 1，所以被用来描述两个量都高的程度.

我们已经知道了，对于一个以概率作为输出的模型，改变阈值，将会影响模型给出的判断，从而影响混淆矩阵上各个元素的值，每改变一次阈值，我们就更新一次召回率，并且更新一次留出率：

$$\text{Fall-out} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{4.7}$$

留出率就是负类中，被漏掉的比例，让阈值等距地取遍 0 到 1 之间的一些数，比如说 $0, 0.1, 0.2, \cdots, 1$，我们可以得到一个表格

表 4-1: ROC 图像坐标值

| criterion | recall | fall-out |
|---|---|---|
| 0 | 1 | 0 |
| 0.1 | ⋮ | ⋮ |
| ⋮ | ⋮ | ⋮ |
| 1 | 0 | 1 |

拿 (fall-out, recall) 作为直角坐标，在二维直角坐标系上描点并连线，就能够得到 ROC 曲线. 假如我们要比较多个模型各自的好坏，可以在同一个坐标系上描出各自的 ROC 曲线，然后观察被曲线和底部与右部直线包围的面积，面积较大者优，因为如果 Recall 和 Fall-out 同时高则表明模型既不漏，也不重. 由于 ROC 曲线的计算是对于以概率值作为输出的模型而言的，所以实际上，要计算一条 ROC 曲线各个点的坐标，模型甚至都不重要，只要知道模型在对于各个样本作为输入所输出的概率就行了：

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn import metrics
4 y = np.array([0, 0, 0, 0, 0, 1, 1, 1])
5 scores = np.array([0.1, 0.3, 0.49, 0.2, 0.4, 0.35, 0.8, 0.6])
6 fpr, tpr, thresholds = metrics.roc_curve(y, scores)
7 auc = metrics.auc(fpr, tpr)
8 plt.figure()
9 lw = 2
```

```
10 plt.plot(
11     fpr,
12     tpr,
13     color='darkorange',
14     lw=lw,
15     label='ROC curve (area = %0.2f)' % auc
16 )
17 plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
18 plt.xlabel('Fall-out')
19 plt.ylabel('Recall')
20 plt.title('ROC-Curves')
21 plt.legend(loc="lower right")
22 plt.show()
```

上列代码中，`y` 可看做是附在样本上的真实标签，`scores` 可看做是模型给出的 `y=1` 的概率，仅仅有了这两样东西，就足够计算出 ROC 曲线了. 图 4-1是画出的 ROC 曲线图像，由于数据量非常少，所以看起来非常简单.

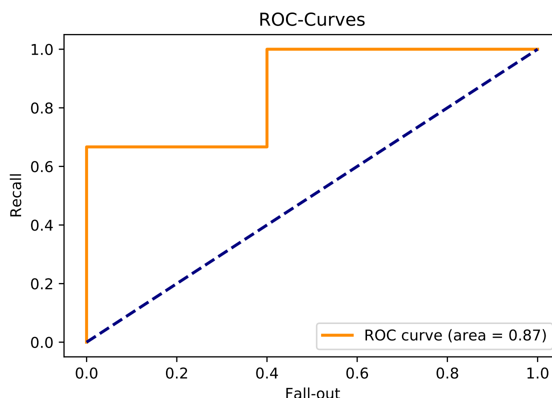


图 4-1: 用少量数据计算出的 ROC 曲线

## 4.2 确定以及肯定的程度

假如说我们已经知道一组真实标签是

```
[ 0, 0, 0, 1, 1, 1, 1 ]
```

然后我们让两个模型 $f_1$ 和 $f_2$ 分别预测样本是正类（也就是标签值为 1）的概率，输出分别为

```
[ 0.10, 0.10, 0.30, 0.98, 0.80, 0.87, 0.97 ]
[ 0.42, 0.30, 0.49, 0.53, 0.61, 0.74, 0.62 ]
```

那么这两个模型，那一个本领更高强呢？毫无疑问是第一个，首先，两个模型的准确率都是 100%，在此前提下，自然是越肯定的，越优秀，而第一个模型对于标签为 0 的类别，输出的概率都比较接近 0，对于标签为 1 的类别，输出的概率都接近 1，表明它对自己的结果非常自信. 而第二个就好像是上课被老师叫起来回答问题，吞吞吐吐且小声回答的学生，虽说碰巧答对了，但是它对于这份答案看似没什么把握.

我们有一个明确的指标来作为这个「确定以及肯定以及答对了」的程度的度量，他就是交叉熵 (cross entropy)，设 $y_i$，$i = 1, 2, \cdots, N$ 是真实标签（取值 0,1），设 $\hat{y}_i$，$i = 1, 2, \cdots, N$ 是模型输出的概率，那么

可以用

$$J(\boldsymbol{\omega}) = -\frac{1}{N}\left(\sum_{i=1}^{N} y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right) \tag{4.8}$$

来衡量这个模型的正确度以及对自己所做预测的信心，具体是，交叉熵的值越低，那么模型的给出的结果就更接近正确同时模型对自己给出的结果的正确性也就越有信心.

简单计算一下两个模型各自的交叉熵表现,

```
1 y0 = np.array([ 0, 0, 0, 1, 1, 1, 1 ])
2 y1 = np.array([ 0.10, 0.10, 0.30, 0.98, 0.80, 0.87, 0.97 ])
3 y2 = np.array([ 0.42, 0.30, 0.49, 0.53, 0.61, 0.74, 0.62 ])
4
5 cross_entropy_1 = -(np.sum(y0 * np.log(y1) + (1-y0) * np.log(1-y1))/y0.shape[0])
6 cross_entropy_2 = -(np.sum(y0 * np.log(y2) + (1-y0) * np.log(1-y2))/y0.shape[0])
7
8 print(cross_entropy_1)
9 print(cross_entropy_2)
```

算得:

```
0.14006621552919005
0.4975803086602629
```

发现：第一个模型的交叉熵更低，第二个模型的交叉熵更高，与实际符合. 由于式 4.8具备的分析性质（它不像 precision 和 recall 那样要数数求混淆矩阵)，所以它再加上一个正则化项就常被用来当做分类任务中的损失函数.