机器学习导论
习题一

171860607, 白晋斌, 810594956@qq.com

2020 年 3 月 15 日

学术诚信

本课程非常重视学术诚信规范，助教老师和助教同学将不遗余力地维护作业中的学术诚信规范的建立。希望所有选课学生能够对此予以重视。[1]

(1) 允许同学之间的相互讨论，但是**署你名字的工作必须由你完成**，不允许直接照搬任何已有的材料，必须独立完成作业的书写过程;

(2) 在完成作业过程中，对他人工作（出版物、互联网资料）中文本的直接照搬（包括原文的直接复制粘贴及语句的简单修改等）都将视为剽窃，剽窃者成绩将被取消。**对于完成作业中有关键作用的公开资料，应予以明显引用**;

(3) 如果发现作业之间高度相似将被判定为互相抄袭行为，**抄袭和被抄袭双方的成绩都将被取消**。因此请主动防止自己的作业被他人抄袭。

作业提交注意事项

(1) 请在 LaTeX 模板中第一页填写个人的姓名、学号、邮箱信息;

(2) 本次作业需提交该 pdf 文件、问题 2 问题 4 可直接运行的源码 (两个.py 文件)、作业 2 用到的数据文件 (为了保证问题 2 代码可以运行)，将以上四个文件压缩成 zip 文件后上传，例如 181221001.zip;

(3) 未按照要求提交作业，或提交作业格式不正确，将会被扣除部分作业分数;

(4) 本次作业提交截止时间为 3 月 15 日 23:59:59。除非有特殊情况（如因病缓交），否则截止时间后不接收作业，本次作业记零分。

[1] 参考尹一通老师高级算法课程中对学术诚信的说明。

Problem 1

若数据包含噪声，则假设空间中有可能不存在与所有训练样本都一致的假设，此时的版本空间是什么？在此情形下，试设计一种归纳偏好用于假设选择。

Solution.

“若数据包含噪声，则假设空间中有可能不存在与所有训练样本都一致的假设.”换而言之, 该噪声即为与训练集中某些样本特征完全一样, 但标签不同的训练样本. 正常情况的版本空间为与训练集一致的“假设集合”, 但此时训练集自身失去一致性, 故版本空间为空集.

一种对样本数据利用率较高的归纳偏好为: 对比这样特征完全相同的样本的标签数量, 保留标签数量最多的一条样本, 扔掉其他样本; 若标签数量最多的样本存在多条时, 扔掉所有样本. 例如, 若这些特征完全相同的样本的标签分别为 A,A,A,B,B,C, 我们保留一条标签为 A 的样本, 扔掉其他五条样本; 若这些特征完全相同的样本的标签分别为 A,B, 我们扔掉所有样本.

Problem 2 [编程]

现有 500 个测试样例,其对应的真实标记和学习器的输出值如表1所示 (完整数据见 data.csv 文件)。该任务是一个二分类任务，1 表示正例，0 表示负例。学习器的输出越接近 1 表明学习器认为该样例越可能是正例，越接近 0 表明学习器认为该样例越可能是负例。

表 1: 测试样例表

样本	x_1	x_2	x_3	x_4	x_5	...	x_{496}	x_{497}	x_{498}	x_{499}	x_{500}
标记	1	1	0	0	0	...	0	1	0	1	1
输出值	0.206	0.662	0.219	0.126	0.450	...	0.184	0.505	0.445	0.994	0.602

(1) 请编程绘制 P-R 曲线

(2) 请编程绘制 ROC 曲线，并计算 AUC

本题需结合关键代码说明思路，并贴上最终绘制的曲线。建议使用 Python 语言编程实现。(预计代码行数小于 100 行)

提示:

- 需要注意数据中存在输出值相同的样例。
- 在 Python 中，数值计算通常使用 Numpy, 表格数据操作通常使用 Pandas, 画图可以使用 Matplotlib (Seaborn), 同学们可以通过上网查找相关资料学习使用这些工具。未来同学们会接触到更多的 Python 扩展库，如集成了众多机器学习方法的 Sklearn, 深度学习工具包 Tensorflow, Pytorch 等。

Solution.

代码如下, 分析见后文.

```python
import pandas as pd
import matplotlib.pyplot as plt
def problem2_1():
    orgData = pd.read_csv("data.csv")
    # 将数据按照output(预测值)降序重排序
    sortData = orgData.sort_values(by=["output"], ascending=False)
    sortData=sortData.reset_index(drop=True)
    T = 0
    F = 0
    #统计原数据中真实情况的正例和反例各有多少
    for i in range(len(sortData)):
        if sortData["label"][i] == 1:
            T += 1
        else:
            F += 1
    #默认全部预测为反例,计算对应的P值与R值得到点对(P,R)
    TP = 0
    FP = 0
    TN = F
    FN = T
    Plist = [1]
    Rlist = [0]
    #按照预测值的高低,逐个将数据预测为正例,重新计算P与R并保存
    for i in range(len(sortData)):
        if sortData["label"][i] == 1:
            TP += 1
            FN -= 1
        else:
            FP += 1
            TN -= 1
        P = TP / (TP + FP)
        R = TP / (TP + FN)
        #如果发现下一条数据预测值与本条相同
        #则本条预测值修正为正例后,暂不更新P,R
        if i<len(sortData)-1:
            if sortData["output"][i] == sortData["output"][i+1]:
                continue
        Plist.append(P)
        Rlist.append(R)
    #最终得到保存P和R的两个list,进行绘图
```

```
    plt.xlabel('recall')
    plt.ylabel('precision')
    plt.plot(Rlist, Plist)
    plt.savefig("P_Rcurve")
    plt.show()

def problem2_2():
    orgData = pd.read_csv("data.csv")
    #将数据按照output(预测值)降序重排序
    sortData = orgData.sort_values(by=["output"], ascending=False)
    sortData = sortData.reset_index(drop=True)
    T = 0
    F = 0
    #统计原数据中真实情况的正例和反例各有多少
    for i in range(len(sortData)):
        if sortData["label"][i] == 1:
            T += 1
        else:
            F += 1
    #默认全部预测为反例,计算对应的TPR值与FPR值得到点对(TPR,FPR)
    TP = 0
    FP = 0
    TN = F
    FN = T
    TPRlist = []
    FPRlist = []
    #按照预测值的高低,逐个将数据预测为正例,重新计算TPR与FPR并保存
    for i in range(len(sortData)):
        if sortData["label"][i] == 1:
            TP += 1
            FN -= 1
        else:
            FP += 1
            TN -= 1
        TPR = TP / (TP + FN)
        FPR = FP / (TN + FP)
        #如果发现下一条数据预测值与本条相同
        #则本条预测值修正为正例后,暂不更新TPR与FPR.
        if i<len(sortData)-1:
            if sortData["output"][i] == sortData["output"][i+1]:
```

```
                    continue
            TPRlist.append(TPR)
            FPRlist.append(FPR)
        #最终得到保存TPR与FPR的两个list,进行绘图
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.plot(FPRlist, TPRlist)
        plt.savefig("ROCcurve")
        plt.show()
        AUC=0
        for i in range(len(TPRlist)-1):
            AUC+=(0.5*(FPRlist[i+1]-FPRlist[i])*(TPRlist[i]+TPRlist[i+1]))
        print("AUC=",AUC)

if __name__ == '__main__':
    problem2_1()
    problem2_2()
```

分析:

(1) R-R 曲线由函数 probelm2_1 实现, 首先是将数据按照 output(预测值) 降序重排序, 之后统计原数据中真实情况的正例和反例各有多少, 默认全部预测为反例, 计算对应的 P 值与 R 值得到点对 (P,R), 之后按照预测值的高低, 逐个将数据预测为正例, 重新计算 P 与 R 并保存. 额外注意的是, 这里用了向下看一步的思路, 如果发现下一条数据预测值与本条相同, 则本条预测值修正为正例后, 暂不更新 P,R. 最终得到保存 P 和 R 的两个 list, 进行绘图, 绘制结果如图1. 结合关键代码的说明详见前文代码注释.

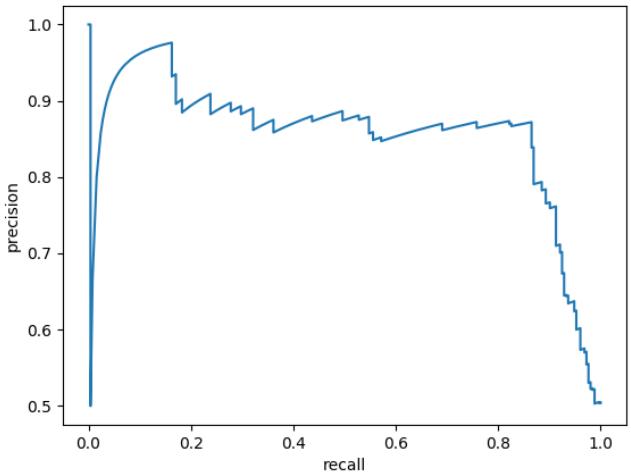

图 1: P-R 曲线图

(2) ROC 曲线由函数 probelm2_2 实现, 首先是将数据按照 output(预测值) 降序重排序, 之后统计原数据中真实情况的正例和反例各有多少, 默认全部预测为反例, 计算对应的 TPR 值与 FPR 值得到点对 (TPR,FPR), 之后按照预测值的高低, 逐个将数据预测为正例, 重新计算 TPR 与 FPR 并保存. 额外注意的是, 这里用了向下看一步的思路, 如果发现下一条数据预测值与本条相同, 则本条预测值修正为正例后, 暂不更新 TPR 与 FPR. 最终得到保存 TPR 与 FPR 的两个 list, 进行绘图, 绘制结果如图2. 结合关键代码的说明详见前文代码注释.

代码最终输出为 AUC= 0.8737.(保留四位小数)

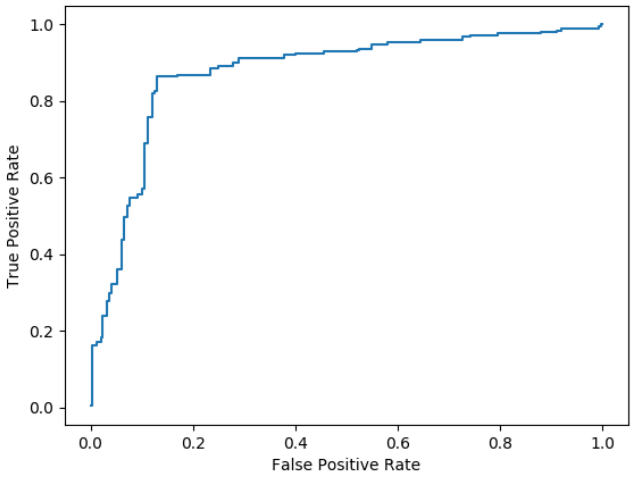

图 2: ROC 曲线图

Problem 3

对于有限样例，请证明

$$\mathrm{AUC} = \frac{1}{m^+m^-}\sum_{x^+\in D^+}\sum_{x^-\in D^-}\left(\mathbb{I}(f(x^+) > f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))\right)$$

Proof.

由 ROC 曲线图可知,ROC 曲线图横轴代表着 FPR, 即 $\frac{TP}{TP+FN}$, 纵轴代表着 TPR, 即 $\frac{FP}{TN+FP}$. 并且由定义知,$TP + FN$ 为真实情况所有正例,$TN + FP$ 为真实情况所有反例. 因此在同一份样本集中,ROC 曲线图横轴与纵轴的分母为定值. 此外, 每当有一个真实情况为正例的样本被正确判断为正例时,TP 增大, 对应 ROC 曲线上的点上移长度为 $\frac{1}{TP+FN}$; 每当有一个真实情况为反例的样本被错误判断为正例时,FP 增大, 对应 ROC 曲线上的点右移 $\frac{1}{TN+FP}$.(这里不妨设每个样本的预测对图形的影响在曲线图中所占为单位距离 1, 初始情况即所有样本被预测为反例对应 (0,0), 最终情况所有样本被预测为正例则表示初始点向上或向右累计一共移动了样本集数量个单位长度)

现在我们证明公式

$$\mathrm{AUC} = \frac{1}{m^+m^-}\sum_{x^+\in D^+}\sum_{x^-\in D^-}\left(\mathbb{I}(f(x^+) > f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))\right)$$

首先, 我们考虑一个真实情况为正例的样本, 即 x_i^+,$\sum_{x^-\in D^-}$ 可以看作是遍历所有真实情况为反例的样本,$\mathbb{I}(f(x^+) > f(x^-))$ 与 $\mathbb{I}(f(x^+) = f(x^-))$ 即对比所有真实情况为反例的样本的预测值与该正例样本的预测值,$\mathbb{I}(f(x^+) > f(x^-))$ 表示当反例预测值小于该正例预测值, 指示函数为 1, 即在绘制 ROC 曲线表时, 正例优先反例被预测为正例, 此指示函数可被认为是先向上的长度为 1 的向量和后向右的长度为 1 的向量组成的图像 (仅考虑该正例与该反例). 同时该指示函数得到的值为 1, 即可认为是该向上向右折线所围成的大小为 1 的方块, 且该方块恰好对应折线正右方或正下方某一块.$\mathbb{I}(f(x^+) = f(x^-))$ 表示当反例预测值等于该正例预测值, 指示函数为 1, 此时正例和反例同时被预测为正例, 此指示函数可被认为是同时向上的长度为 1 的向量和向右的长度为 1 的向量, 即指向右上方方向 45° 的向量组成的图像,$\mathbb{I}(f(x^+) = f(x^-))$ 得到的值为 1, 但指向右上方方向 45° 向量将一个小方块分割成上下两块, 面积分别为 $\frac{1}{2}$, 对应 $\frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))$ 即可代表该右上方方向 45° 的向量对应正下方的三角形区域. 因为只有 1 个正例样本, 所以 ROC 图可看作是纵轴长度为 1 的特殊图, 图中包含向上向后向右上 45° 三种向量, 该向量与 ROC 图的下方和右方共同围成了面积为 $\sum_{x^-\in D^-}\left(\mathbb{I}(f(x^+) > f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))\right)$ 的图形.

$\sum_{x^+\in D^+}$ 可以看作是考虑所有真实情况为正例的样本. 即将 ROC 图扩展成纵轴长度为正例样本数量的图, 此时 $\sum_{x^+\in D^+}$ 可看作是对所有纵轴长度为 1 的特殊图的累加, 最终得到了整个 ROC 曲线右下方所包围的面积.

但是, 此时我们得到的 ROC 图横轴长度为反例样本数 m^-, 纵轴为正例样本数 m^+, 而标准 ROC 图的横轴纵轴都是单位长度 1, 故我们乘系数 $\frac{1}{m^+m^-}$, 从而得到

$$\mathrm{AUC} = \frac{1}{m^+m^-}\sum_{x^+\in D^+}\sum_{x^-\in D^-}\left(\mathbb{I}(f(x^+) > f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-))\right)$$

注: 可在初始表述时严格认为每当有一个真实情况为正例的样本被正确判断为正例时,TP 增大, 对应 ROC 曲线上的点上移长度为 $\frac{1}{TP+FN}$ 即 $\frac{1}{m^+}$, 相当于将系数放置于求和符号内. 但我们为便于表述, 将每个样本对图像的影响视作单位长度 1, 如第一段括号内所述, 最终再对 ROC 图做归一化. □

Problem 4 [编程]

在数据集 D_1, D_2, D_3, D_4, D_5 运行了 A, B, C, D, E 五种算法，算法比较序值表如表2所示:

表 2: 算法比较序值表

数据集	算法 A	算法 B	算法 C	算法 D	算法 E
D_1	2	3	1	5	4
D_2	5	4	2	3	1
D_3	4	5	1	2	3
D_4	2	3	1	5	4
D_5	3	4	1	5	2
平均序值	3.2	3.8	1.2	4	2.8

使用 Friedman 检验 ($\alpha = 0.05$) 判断这些算法是否性能都相同。若不相同，进行 Nemenyi 后

续检验 ($\alpha = 0.05$)，并说明性能最好的算法与哪些算法有显著差别。本题需编程实现 Friedman 检验和 Nemenyi 后续检验。(预计代码行数小于 50 行)

Solution.

代码如下, 分析在代码后面.

```
import math
import matplotlib.pyplot as plt

def Friedman(performance,avg):
    N = len(performance)
    k = len(performance[0])
    a = (12 * N) / (k * (k + 1))
    sum = 0
    for i in avg:
        sum += (i * i)
    sum -= (k * (k + 1) * (k + 1) / 4)
    Tx2 = a * sum
    TF = ((N - 1) * Tx2) / (N * (k - 1) - Tx2)
    print("Friedman:",TF)

def Nemenyi(performance,avg):
    N = len(performance)
    k = len(performance[0])
    qa=2.728
    CD=qa*math.sqrt((k*(k+1))/(6*N))
    print("Nemenyi:",CD)
    return CD

if __name__ == '__main__':
    performance = [[2, 3, 1, 5, 4],
                   [5, 4, 2, 3, 1],
                   [4, 5, 1, 2, 3],
                   [2, 3, 1, 5, 4],
                   [3, 4, 1, 5, 2]]
    avg = [3.2, 3.8, 1.2, 4, 2.8]
    avg_name=["alg_A","alg_B","alg_C","alg_D","alg_E"]
    Friedman(performance, avg)
    CD=Nemenyi(performance,avg)
    plt.scatter(avg,avg_name)
    algmin=[]
    algmax=[]
```

```
37      for i in avg:
38          algmin.append(i-CD/2)
39          algmax.append(i+CD/2)
40      plt.hlines(avg_name,algmin,algmax)
41      plt.savefig("Friedmanfig")
42      plt.show()
```

分析:

首先使用 Friedman 检验 (Friedman 函数) 计算出 $\tau_F = 3.937$, 查表知其大于 $\alpha = 0.05$ 时的 F 检验临界值 3.007, 因此拒绝"所有算法性能相同"这个假设. 然后使用 Nemenyi 后续检验 (Nemenyi 函数), 查表得 $\alpha = 0.05, k = 5$ 时的 $q_\alpha = 2.728$, 带入 Nemenyi 函数, 计算出临界值域 $CD = 2.728$, 由题中所给的平均序值可知, 仅有算法 C 与算法 D 的差距超过临界值域, 其他算法两两之间均未超过, 因此检验结果仍为算法 C 与 D 的性能显著不同, 而其他算法两两之间的性能并没有显著差别.

上述检验比较可以直观地用 Friedman 检验图显示. 从图3中可以容易地看出, 性能最好的算法是 C, 其与算法 D 有显著差别, 与算法 A,B,E 无显著差别.

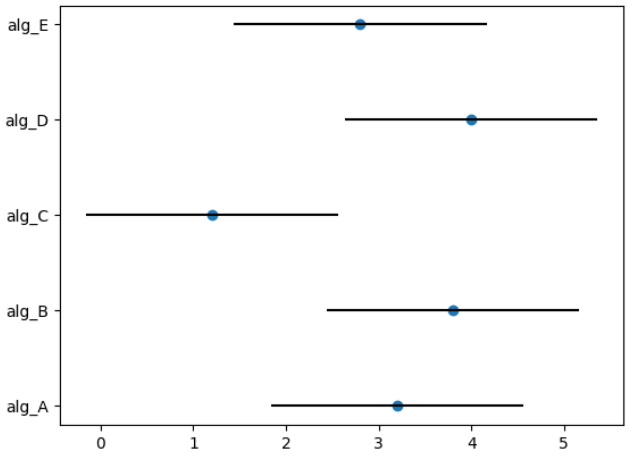

图 3: Friedman 检验图