CS221 Fall 2021 Homework——Foundations

Zheng Yijie, 19211416

2022 年 5 月 24 日

1 优化和概率

1.1 Optimize weighted average

***Q*: 求 $f(\theta)=\sum_{i=1}^{n} w_i\left(\theta-x_i\right)^2$ 的最小值点。当 w_i 为负数时会怎样？**

$f(\theta)$ 在极小值点处的一阶导数应为 0 且二阶导数应大于 0。

$$\begin{aligned} f'(\theta)=\frac{df(\theta)}{d\theta}=\sum_{i=1}^{n} w_i\left(\theta-x_i\right) \\ f''(\theta)=\frac{df'(\theta)}{d\theta}=\sum_{i=1}^{n} w_i \end{aligned} \tag{1}$$

令 $f'(\theta)=0$，则

$$\theta=\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \tag{2}$$

由于 $w_i>0$，$f''(\theta)>0$ ，此时 $\theta=\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$ 为极小值点。若 w_i 不全大于 0，则 $f''(\theta)$ 可能小于等于 0，此时 $f(\theta)$ 不存在最小值。

1.2 Swap sum and max

***Q*: 比较 $f(\mathbf{x})=\max_{s\in[-1,1]}\sum_{i=1}^{d} s x_i$ 与 $g(\mathbf{x})=\sum_{i=1}^{d}\max_{s_i\in[-1,1]} s_i x_i$ 的大小**

$f(\mathbf{x})\leq g(\mathbf{x})$ 。证明:

$$\begin{aligned} g(x)&=\sum_{i=1}^{d}\max_{s_i\in[1,-1]} s_i x_i \\ &=\sum_{i=1}^{d}|x_i| \end{aligned} \tag{3}$$

其中，$s_i=sgn(x_i)$

$$\begin{aligned} f(x)&=\max_{s\in[1,-1]}\sum_{i=1}^{d} s x_i \\ &\leq\sum_{i=1}^{d}|x_i| \\ &=g(x) \end{aligned} \tag{4}$$

当且仅当 x_i 同号时等号成立。证毕。

1.3 Expected value of iterated game

***Q*: 求骰子游戏的得分期望**

状态转移:

- 投到 1 或 2，停止
- 投到 3，得分减 a
- 投到 6，得分加 b
- 投到 4 或 5，得分不变

设 V 为得分期望。只考虑投到 1，2，3，6 的情况（其余情况不影响得分）

$$V = \frac{1}{2} \times 0 + \frac{1}{4} \times (V - a) + \frac{1}{4} \times (V + b) \tag{5}$$

解得

$$V = \frac{b - a}{2} \tag{6}$$

1.4 Derive maximum likelihood

***Q*: 求硬币正面朝上的概率，使得抛六次硬币结果为 *{T,H,H,H,T,H}* 的概率最大**

抛六次硬币结果为 {T,H,H,H,T,H} 的概率为 $L(p) = (1-p)ppp(1-p)p = p^4(1-p)^2$。问题等价于求 $\arg\, max_p L(p)$，也即 $\arg\, max_p \ln L(p)$ 。

$$\begin{aligned} \ln L(p) &= 4\ln p + 2\ln(1-p) \\ \frac{d\ln L(p)}{dp} &= \frac{4}{p} - \frac{2}{1-p} = 0 \\ 0 &= 4 - 6p \\ p &= \frac{2}{3} \end{aligned} \tag{7}$$

为验证 $p = \frac{2}{3}$ 为极大值点，计算 $L(p)$ 在 $p = \frac{2}{3}$ 处的二阶导数。

$$\frac{d^2 L(p)}{dp^2}|_{p=\frac{2}{3}} = [12p^2(1-p)^2 - 16p^3(1-p) + 2p^4]_{p=\frac{2}{3}} = -0.59 < 0 \tag{8}$$

因此 $p = \frac{2}{3}$ 时，抛六次硬币结果为 {T,H,H,H,T,H} 的概率最大。

1.5 Manipulate conditional probabilities

***Q*: 已知** $P(A \cup B) = P(A) + P(B) - P(A \cap B) = \frac{1}{2}, P(A \cap B) > 0, P(A|B) = P(B|A)$ **证明** $P(A) > \frac{1}{4}$

$$\because \begin{cases} P(A \cap B) = P(AB) = P(A)P(B|A) = P(B)P(A|B) \\ P(A|B) = P(B|A) \end{cases} \tag{9}$$

$$\therefore P(A) = P(B) \tag{10}$$

$$\because \begin{cases} P(A\cup B)=P(A)+P(B)-P(A\cap B)=\frac{1}{2} \\ P(A\cap B)>0 \end{cases} \tag{11}$$

$$\therefore 2P(A)=P(A)+P(B)=P(A\cup B)+P(A\cap B)>\frac{1}{2} \tag{12}$$

$$\therefore P(A)>\frac{1}{4} \tag{13}$$

1.6 Take gradient

***Q*: 求 $f(\mathbf{w})=\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{a}_i^\top\mathbf{w}-\mathbf{b}_j^\top\mathbf{w}\right)^2\right)+\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ 的梯度 $\nabla f(\mathbf{w})$**

令 $a_i=(a_{i1},a_{i2},\dots a_{id})^\top$， $b_j=(b_{j1},b_{j2},\dots b_{jd})^\top$ ， 则

$$\begin{aligned} f(w)&=\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_i^\top w-b_j^\top w\right)^2+\frac{\lambda}{2}\|w\|_2^2 \\ &=\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_{i1}w_1+a_{i2}w_2\dots a_{id}w_{id}-(b_{j1}w_1+b_{j2}w_2\dots b_{jd}w_{jd})\right)^2+\frac{\lambda}{2}\sqrt{\sum_{k=1}^{d}w_k^2} \end{aligned} \tag{14}$$

$$\begin{aligned} \nabla f(w)&=\left(\frac{\partial f(w)}{\partial w_1},\dots\frac{\partial f(w)}{\partial w_d}\right)^\top \\ &=\left(2\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_i^\top w-b_j^\top w\right)(a_{i1}-b_{j1})+\lambda\frac{w_1}{||w||_2^2},\dots,2\sum_{i=1}^{n}\sum_{j=1}^{n}\left(a_i^\top w-b_j^\top w\right)(a_{id}-b_{jd})+\lambda\frac{w_1}{||w||_2^2}\right)^\top \end{aligned} \tag{15}$$

2 复杂度

2.1 Counting rectangles

***Q*: 求在 $n\times n$ 的网格图中放置 *4* 个矩形的可能方式的复杂度**

1 个矩形可以由左下角和右上角的两个顶点的坐标 $(x_1,y_1),(x_2,y_2)$ 确定。约束是 $x_1\le x_2,y_1\le y_2$ ， $x,y\in\{0,1,2,...,n\}$ ， 横纵坐标相互独立。因此单个矩形的位置数 $C1=[n+1+\binom{n}{2}]^2=(\frac{n(n+1)}{2})^2$, 复杂度为 $O(n^4)$ 。

所以 4 个矩形的可能位置数为在单个矩形的位置空间中寻找 4 个元素的任意组合。可能的组合数为 $C4=\binom{C1}{4}+C1$, 复杂度为 $O(n^{4\times4}+4)=O(n^{16})$

2.2 Dynamic program

***Q*: 在 $n\times3n$ 网格图中，已知经过每一个网格的定代价函数 $c(i,j)$，每次只能向右或向下移动一个网格，求从左上角网格 $(1,1)$ 到右下角网格 $(n,3n)$ 的最小代价，并且算法的时间复杂度最小。**

设 $c_{min}(i,j)$ 为每一个网格的最小累积代价。由于移动方向只能是向右或向下，每一个网格的最小累积代价由其左边和上面相邻网格代价的累积代价最小值决定，即

$$c_{\min}(i,j) = c(i,j) + \min_{(i,j)\in\{(i-1,j),(i,j-1)\}} c_{\min}(i,j) \tag{16}$$

搜索时，先计算第 1 行和第 1 列（为了便于理解，数组下标从 1 开始）的最小累计代价，再使用两层循环计算剩余网格的最小累计代价，所以算法复杂度为 $O(3n^2)$。

Algorithm 1: 最小代价搜索

Data: 网格大小 $n \times 3n$，单步代价矩阵 cost[n][3n]

Result: 最小累计代价矩阵 min_cost[n][3n]

```
min_cost[1][1]←cost[1][1]
计算第一列的最小累计代价
for i = 2 to n do
    min_cost[i][1←cost[i][1] + min_cost[i-1][1]
end
计算第一行的最小累计代价
for i = 2 to 3n do
    min_cost[1][i] ← cost[1][i] + min_cost[1][i-1]
end
计算剩余网格的最小累计代价
for i = 2 to n do
    for j = 2to 3n do
        min_cost[1][i] ← cost[i][j] + min(min_cost[i-1][j],min_cost[i][j-1])
    end
end
```

3 AI 伦理问题

在这个问题中，您将使用机器学习 NeurIPS 会议制定的道德准则探索四种不同现实世界场景的道德规范。NeurIPS 道德准则在潜在的负面社会影响和一般道德行为（编号列表）下列出了 16 个非详尽的问题。对于每种场景，你都将编写一份潜在的负面影响声明。为此，你将首先确定算法/数据集/技术是否可能产生潜在的负面社会影响或违反一般道德行为（再次，从 NeurIPS 道德指南页面中获取的十六个编号项目）。如果场景确实违反了道德行为或具有潜在的负面社会影响，请列出它所违反的一个问题，并说明您认为该问题适用于该场景的原因。如果您认为该场景不存在道德问题，请解释您是如何做出该决定的。与以往的问题不同，本问题有许多可能的优质答案。如果你能证明你的答案是正确的，那么你应该确信你已经很好地回答了这个问题。

AI 道德潜在的负面社会影响，一般道德行为和数据相关问题考虑：

- 潜在负面社会影响方面，是否有以下可能：

1. 直接威胁生命
2. 安全漏洞问题
3. 人权问题
4. 民生和社会经济安全
5. 发展有害的监控行为
6. 环境破坏
7. 欺骗他人并造成伤害

- 一般道德行为方面，应该考虑数据集是否可能：

1. 包含个人隐私
2. 包含可以推断出他人未同意共享的信息
3. 数据集对特定性别、种族、性取向或具有其他特征的人的偏见
4. 包含人类受试者实验，是否已被相关监督委员会批准
5. 被数据集发布者删除

- 并考虑和审查其他和数据相关的问题：

1. 同意使用或共享数据
2. 与高风险群体合作时的注意事项
3. 过滤具有冒犯性的内容
4. 遵守 GDPR 和其他数据相关法规

3.1 贷款违约风险

***Q*：一家投资公司开发了一个简单的机器学习模型来预测个人是否有可能从各种因素（包括地点、年龄、信用评分和公共记录）中拖欠贷款。查看他们的结果后，您会发现该模型主要基于位置进行预测，并且该模型主要接受来自城市中心的贷款并拒绝来自农村申请人的贷款。此外，查看申请人的性别和种族，您会发现该模型对黑人和男性申请人的误报率明显高于其他群体。**

由于该算法可以预测拖欠带宽的概率，因此该算法可能已经编码、包含或可能加剧对某个种族或民族的人的偏见。这可能会导致人权问题——黑人、男性、来自农村的三类人群在申请贷款上收到歧视。

3.2 去匿名算法

***Q*：*Stylometry* 是一种通过分析匿名文本和潜在作者撰写的其他文本中的写作模式来预测有争议或匿名文本作者的方法。最近，已经为这项任务开发了高精度的机器学习算法。虽然这些模型通常用于分析历史文献和文献，但它们可用于对包括代码在内的各种文本进行去匿名化。**

这项技术可以根据已有信息推断出他人不愿意公开的信息，有侵犯个人隐私的风险，可能会使一些匿名作者受到伤害。

3.3 名人面部数据集

***Q*: 一个研究小组从谷歌图像中抓取了数百万张名人的面孔，以开发面部识别技术。名人没有允许他们的图像在数据集中使用，并且许多图像都受版权保护。对于受版权保护的照片，数据集提供原始图像的 *URL* 链接以及面部的边界框。**

未经过他人同意、无视版权收集其名人面孔并公开，数据集的非法使用和传播可能会侵犯名人的隐私权和肖像权。

3.4 植物识别

***Q*: 研究人员最近创建了一种机器学习模型，可以直接从单张照片中自动预测植物物种。该模型是使用同意将照片用于研究目的的用户上传到 *iNaturalist* 应用程序的照片进行训练的，并且该模型仅在应用程序中用于帮助用户识别他们可能在野外遇到的植物。**

数据集来源经用户同意，研究成果仅能用于植物种类识别，不会对社会造成潜在危害，也不存在一般道德问题。

4 编程

代码地址https://github.com/VoyagerXvoyagerx/AIFoundations

讲解视频地址https://space.bilibili.com/475685954

4.1 Find alphabetically first word

```
# Problem 4a

def find_alphabetically_first_word(text: str) -> str:
    """
    Given a string |text|, return the word in |text| that comes first
    lexicographically (i.e., the word that would come first after sorting).
    A word is defined by a maximal sequence of characters without whitespaces.
    You might find max() handy here. If the input text is an empty string,
    it is acceptable to either return an empty string or throw an error.
    """
    # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
    return min(text.lower().split())
    # END_YOUR_CODE
```

4.2 Euclidean distance

```
# Problem 4b

def euclidean_distance(loc1: Position, loc2: Position) -> float:
```

```
    """
    Return the Euclidean distance between two locations, where the locations
    are pairs of numbers (e.g., (3, 5)).
    """
    # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
    return math.sqrt((loc1[0]-loc2[0])**2 + (loc1[1]-loc2[1])**2)
    # END_YOUR_CODE
```

4.3 Mutate sentences

```
# Problem 4c

def mutate_sentences(sentence: str) -> List[str]:
    """
    Given a sentence (sequence of words), return a list of all "similar"
    sentences.
    We define a sentence to be "similar" to the original sentence if
      - it has the same number of words, and
      - each pair of adjacent words in the new sentence also occurs in the
        original sentence (the words within each pair should appear in the same
        order in the output sentence as they did in the original sentence).
    Notes:
      - The order of the sentences you output doesn't matter.
      - You must not output duplicates.
      - Your generated sentence can use a word in the original sentence more
        than once.
    Example:
      - Input: 'the cat and the mouse'
      - Output: ['and the cat and the', 'the cat and the mouse',
                 'the cat and the cat', 'cat and the cat and']
                (Reordered versions of this list are allowed.)
    """
    # BEGIN_YOUR_CODE (our solution is 17 lines of code, but don't worry if you deviate from this)
    def appended_options(a, options):
        return [a + [option] for option in options]
    words = sentence.split()
    # 找出每个单词后面的出现的单词
    next_options_map = {}
    for index, word in enumerate(words):
        if word not in next_options_map: # 添加关键字
            next_options_map[word] = []
        if index + 1 < len(words): # 如果不是最后一个单词，添加值
            next_options_map[word].append(words[index + 1])
    # @next_options_map, {'the': ['cat', 'mouse'], 'cat': ['and'], 'and': ['the'], 'mouse': []}
```

```
    temp = []
    results = []
    # 遍历字典中的关键字，找出所有可能的相似句
    for key in next_options_map:
        temp = []
        temp.append([key])
        i = 0 # 可添加单词的最大序号
        while i < len(temp):
            if len(temp[i]) < len(words) and len(next_options_map[temp[i][-1]]) > 0:
                # 对每一个结果的最后一个单词，查找字典
                temp += appended_options(temp[i],
                                         next_options_map[temp[i][-1]])
                del temp[i]
            else:
                i += 1
        results += list(set([' '.join(result)
                             for result in temp if len(result) == len(words)]))
    return list(set([''.join(result) for result in results]))
    # END_YOUR_CODE
```

5 Sparse Vector Dot Product

```
# Problem 4d

def sparse_vector_dot_product(v1: SparseVector, v2: SparseVector) -> float:
    """
    Given two sparse vectors (vectors where most of the elements are zeros)
    |v1| and |v2|, each represented as collections.defaultdict(float), return
    their dot product.

    You might find it useful to use sum() and a list comprehension.
    This function will be useful later for linear classifiers.
    Note: A sparse vector has most of its entries as 0.
    """
    # {'a': 5}, {'b': 2, 'a': 3}
    # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
    return sum(value*v2[index] for index, value in v1.items())
    # END_YOUR_CODE
```

5.1 Increment sparse vector

```
# Problem 4e
```

```
def increment_sparse_vector(v1: SparseVector, scale: float, v2: SparseVector,) -> None:
    """
    Given two sparse vectors |v1| and |v2|, perform v1 += scale * v2.
    If the scale is zero, you are allowed to modify v1 to include any
    additional keys in v2, or just not add the new keys at all.

    NOTE: This function should MODIFY v1 in-place, but not return it.
    Do not modify v2 in your implementation.
    This function will be useful later for linear classifiers.
    """
    # BEGIN_YOUR_CODE (our solution is 2 lines of code, but don't worry if you deviate from this)
    v = collections.defaultdict(float, [(index, value + v1[index]) for index, value in [
                            (index, value*scale) for index, value in v2.items()]])
    v1.update(v)
    # END_YOUR_CODE
```

5.2 Find Nonsingleton Words

```
def find_nonsingleton_words(text: str) -> Set[str]:
    """
    Split the string |text| by whitespace and return the set of words that
    occur more than once.
    You might find it useful to use collections.defaultdict(int).
    """
    # BEGIN_YOUR_CODE (our solution is 4 lines of code, but don't worry if you deviate from this)
    count_map = collections.defaultdict(int)
    for word in text.split():
        count_map[word] += 1
    return set([v for v in count_map if count_map[v] > 1])
    # END_YOUR_CODE
```

6 总结

数学和算法是人工智能的基础!