目录

第十七篇：2022 年 4 月 10 日百度机器学习方向暑期实习面试题 6 道

1、介绍下 SVM 算法

是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机。

SVM 可分为三种：

线性可分 SVM

当训练数据线性可分时，通过最大化硬间隔（hard margin）可以学习得到一个线性分类器，即硬间隔 SVM。

线性 SVM

当训练数据不能线性可分但是近似线性可分时，通过最大化软间隔（soft margin）也可以学习到一个线性分类器，即软间隔 SVM。

非线性 SVM

当训练数据线性不可分时，通过使用核技巧（kernel trick）和最大化软间隔，可以学习到一个非线性 SVM。

SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。

SVM 如何选择核函数

Linear 核：主要用于线性可分的情形。参数少，速度快，对于一般数据，分类效果已经很理想了。

RBF 核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数。有很多人是通过训练数据的交叉验证来寻找合适的参数，不过这个过程比较耗时。

2、介绍下逻辑回归算法

逻辑回归是在数据服从伯努利分布的假设下，通过极大似然的方法，运用梯度下降法来求解参数，从而达到将数据二分类的目的。

逻辑回归的优缺点

优点：

形式简单，模型的可解释性非常好。从特征的权重可以看到不同的特征对最后结果的影响，某个特征的权重值比较高，那么这个特征最后对结果的影响会比较大。

模型效果不错。在工程上是可以接受的（作为 baseline），如果特征工程做的好，效果不会太差，并且特征工程可以并行开发，大大加快开发的速度。

训练速度较快。分类的时候，计算量仅仅只和特征的数目相关。

缺点：

准确率欠佳。因为形式非常的简单，而现实中的数据非常复杂，因此，很难达到很高的准确性。很难处理数据不平衡的问题。

3、介绍下决策树算法

常见的决策树算法有三种：ID3、C4.5、CART 树

ID3 算法的核心是在决策树的每个节点上应用信息增益准则选择特征，递归地构架决策树。

C4.5 算法的核心是在生成过程中用信息增益比来选择特征。

CART 树算法的核心是在生成过程中用基尼指数来选择特征。

4、用通俗的语言介绍下强化学习（Reinforcement Learning）

监督学习的特点是有一个“老师”来“监督”我们，告诉我们正确的结果是什么。在我们在小的时候，会有老师来教我们，本质上监督学习是一种知识的传递，但不能发现新的知识。对于人类整体而言，真正（甚至唯一）的知识来源是实践——也就是强化学习。比如神农尝百草，最早人类并不知道哪些草能治病，但是通过尝试，就能学到新的知识。

学习与决策者被称为智能体，与智能体交互的部分则称为环境。智能体与环境不断进行交互，具体而言，这一交互的过程可以看做是多个时刻，每一时刻，智能体根据环境的状态，依据一定的策略选择一个动作（这里的策略指的是从环境状态到智能体动作或者动作概率之间的映射），然后环境依据一定的状态转移概率转移到下一个状态，与此同时根据此时状态的好坏反馈给智能体一个奖励。智能体可以根据环境的反馈调整其策略，然后继续在环境中探索，最终学习到一个能够获得最多奖励的最优策略。

5、Leetcode34 在排序数组中查找元素的第一个和最后一个位置

```
1. class Solution(object):
2.     def searchRange(self, nums, target):
3.         """
```

```
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        n = len(nums)
        left, right = 0, n - 1
        while left <= right:
            mid = left + (right - left) // 2
            if target == nums[mid]:
                p = q = mid
                while p >= 0 and nums[p] == target:
                    p -= 1
                while q < n and nums[q] == target:
                    q += 1
                return [p + 1, q - 1]
            if target > nums[mid]:
                left = mid + 1
            else:
                right = mid -1
        return [-1,-1]
```

6、Letcode102 层序遍历

思路：BFS

1、如果 root 为空，直接返回 [ ]

2、定义一个数组 queue，并将 root 添加到 queue 中，再定义一个 res 数组保存结果

3、遍历 当前层 queue 的每个左子节点，右子节点，入队列，且要把 queue 更新成当前层的孩子节点列表，直到 queue 为空。

代码如下：

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        queue = [root]
        res = []
        while queue:
            res.append([node.val for node in queue])
            l1 = []
            for node in queue:
                if node.left:
                    l1.append(node.left)
```

```
                if node.right:
                    l1.append(node.right)
            queue = l1
        return res
```

第十八篇：2022 年 4 月 18 日字节跳动机器学习 AILab 一面面试题 6 道

1、BN 和 LN 区别

Batch Normalization 是对这批样本的同一维度特征做归一化， Layer Normalization 是对这单个样本的所有维度特征做归一化。

区别：LN 中同层神经元输入拥有相同的均值和方差，不同的输入样本有不同的均值和方差；BN 中则针对不同神经元输入计算均值和方差，同一个 batch 中的输入拥有相同的均值和方差。所以，LN 不依赖于 batch 的大小和输入 sequence 的长度，因此可以用于 batchsize 为 1 和 RNN 中 sequence 的 normalize 操作。

2、讲讲 self attention

Self Attention 与传统的 Attention 机制非常的不同：传统的 Attention 是基于 source 端和 target 端的隐变量（hidden state）计算 Attention 的，得到的结果是源端的每个词与目标端每个词之间的依赖关系。但 Self Attention 不同，它分别在 source 端和 target 端进行，仅与 source input 或者 target input 自身相关的 Self Attention，捕捉 source 端或 target 端自身的词与词之间的依赖关系；然后再把 source 端的得到的 self Attention 加入到 target 端得到的 Attention 中，捕捉 source 端和 target 端词与词之间的依赖关系。因此，self Attention Attention 比传统的 Attention mechanism 效果要好，主要原因之一是，传统的 Attention 机制忽略了源端或目标端句子中词与词之间的依赖关系，相对比，self Attention 可以不仅可以得到源端与目标端词与词之间的依赖关系，同时还可以有效获取源端或目标端自身词与词之间的依赖关系。

3、Bert 的预训练过程

Bert 的预训练主要包含两个任务，MLM 和 NSP，Masked Language Model 任务可以理解为完形填空，随机 mask 每一个句子中 15%的词，用其上下文来做预测；Next Sentence Prediction 任务选择一些句子对 A 与 B，其中 50%的数据 B 是 A 的下一条句子，剩余 50%的数据 B 是语料库中随机选择的，学习其中的相关性。BERT 预训练阶段实际上是将上述两个任务结合起来，同时进行，然后将所有的 Loss 相加。

4、Pre Norm 与 Post Norm 的区别?

参考：https://kexue.fm/archives/9009

在同一设置下，Pre Norm（也就是 Norm and add）的效果是要优于 Post Norm（Add and Norm）的，但是单独调整的话，Post Norm 的效果是更好的，Pre Norm 结构无形地增加了模型的宽度而降低了模

型的深度，Post Norm 每 Norm 一次就削弱一次恒等分支的权重，所以 Post Norm 反而是更突出残差分支的。

参考：https://zhuanlan.zhihu.com/p/474988236

post-norm 在残差之后做归一化，对参数正则化的效果更强，进而模型的鲁棒性也会更好；pre-norm 相对于 post-norm，因为有一部分参数直接加在了后面，不需要对这部分参数进行正则化，正好可以防止模型的梯度爆炸或者梯度消失，因此，如果层数少 post-norm 的效果其实要好一些，如果要把层数加大，为了保证模型的训练，pre-norm 显然更好一些。

5、GPT 与 Bert 的区别

1） GPT 是单向模型，无法利用上下文信息，只能利用上文；而 BERT 是双向模型。

2） GPT 是基于自回归模型，可以应用在 NLU 和 NLG 两大任务，而原生的 BERT 采用的基于自编码模型，只能完成 NLU 任务，无法直接应用在文本生成上面。

6、如何加速 Bert 模型的训练

BERT 基线模型的训练使用 Adam with weight decay（Adam 优化器的变体）作为优化器，LAMB 是一款通用优化器，它适用于小批量和大批量，且除了学习率以外其他超参数均无需调整。LAMB 优化器支持自适应元素级更新（adaptive element-wise updating）和准确的逐层修正（layer-wise correction）。LAMB 可将 BERT 预训练的批量大小扩展到 64K，且不会造成准确率损失，76 分钟就可以完成 BERT 的训练。

第十九篇：2022 年 4 月 8 日字节跳动机抖音 APP 推荐实习面试题 8 道

1、AUC 是什么？如何计算 AUC?

AUC：随机取一个正样本和一个负样本，正样本的预测值大于负样本预测值的概率。

AUC 计算的关键是找到所有正样本预测值大于负样本预测值的正负样本对。

首先，需要将样本按照预测值进行从小到大排序（最小 score 对应的 sample 的 rank 为 1，第二小 score 对应 sample 的 rank 为 2，以此类推）；

其次，把所有的正类样本的 rank 相加，再减去两个正样本组合的情况。

$$AUC = \frac{\sum_{i \in \text{positiveClass}} \text{rank}_i - \frac{M(1+M)}{2}}{M \times N}$$

2、AUC 线上线下不一致怎么办

线上线下效果不一致，大概率是由线上线下预估环境不一致引起。 预估环境，一般涉及 2 个要素：模型和特征。

模型是否一致

主要包括校验离线模型格式转换、serving 部署，线上模型加载、预估等接口是否有问题

• 特征是否一致

准确是指，线上线下喂给模型的特征是否一致。

与模型一致性检验一样，首先需要校验线上线下特征处理逻辑是否一致等；

其次，与线上真实预估环境相比，离线环境更容易获取到特征，当离线使用线上获取不到的特征时，就会造成离线效果虚高的假象。

严重点的，如特征穿越，即特征中包含标签信息，会造成训练和评估时数据泄露，导致离线评估时 AUC 虚高；轻一点的，如离线使用的特征比线上实时性高，同样会导致线上效果不符合预期。

这就要求离线阶段构造样本时，需要参考线上真实预估环境获取特征时的延迟，通过严格控制离线特征拼接时的咬合时间，保证线下线下喂给模型的特征的一致性。

更好地做法是，落地线上特征日志，直接用于离线训练。

AUC 指标不能有效刻画模型表现。可以尝试 GAUC 代替。

3、召回阶段的负采样是怎么做的?

第一版 i2i 在 batch 内随机负采样，并加了 bias 进行修正；第二版 u2i 借鉴 w2v，以 uv 曝光为权重，全局负采样，但是由于全局负彩成本太大，做了二次哈希，根据样本 uv 进行了分桶；

加 bias 有什么用?

答：负采样的本质是想打压高曝光 item，但是由于高曝光的 item 本身就频繁出现在样本中，随机采可能会打压过头，因此加一个 bias tower 进行修正（其实就是学一个值取平衡正负样本）。

带权负采样的逻辑?

答：如果按照 uv 曝光进行带权负采样，就是先按照 uv 曝光排序，之后累加，最后生成一个随机数，看着随机数落到哪两个 item 的 uv 累加和（前缀和）区间，就采样哪个 item。

4、FM，DeepFM 跟 FFM 的对比

FM 主要解决了 LR 不能主动学习特征交叉组合问题；解决特征稀疏，特征交叉，权重矩阵稀疏学习困难问题；解决特征交叉组合，模型参数量过大，复杂度高问题。

FFM 相比 FM 基础上，引入了 Field-aware，每个特征在每个 field 上有对应不同的隐向量。

DeepFM 是一个端到端的学习模型，DeepFM 将 FM 的隐向量 V VV 同时作为 Deep 的词向量参数，两者共享，让模型自动去学习低阶与高阶的特征交互。

5、手撕 FM 的训练过程

```
def FM_function_L2_Adagrad(dataMatrix, classLabels, k, iter):
    lamda = 1 #正则化参数
    m, n = shape(dataMatrix)
    alpha = 1
    #1、初始化参数
    w = zeros((n, 1))
    w_0 = 0.
    v = normalvariate(0, 0.2) * ones((n, k))
    w0_ada = 1.
    w_ada = 1.
    v_ada = 1.
    w0_grad = 0.
    w_grad = 0.
    v_grad = 0.
    alpha_w0 = alpha
    alpha_w = alpha
```

```
    alpha_v = alpha

    #2、训练
    for it in range(iter):
        for x in range(m):
            inter_1 = dataMatrix[x] * v
            inter_2 = multiply(dataMatrix[x], dataMatrix[x]) * multiply(v, v)
            #完成交叉项
            interaction = sum(multiply(inter_1, inter_1) - inter_2) / 2.
            p = w_0 + dataMatrix[x] * w + interaction
            loss = classLabels[x] * p[0, 0] - 1
            #加入 adagrad
            w0_grad += (loss* classLabels[x] +w_0*lamda) ** 2
            w_0 -= alpha_w0 * (loss* classLabels[x] *+w_0*lamda)
            for i in range(n):
                if dataMatrix[x, i] != 0:
                    #加入 adagrad
                    w_grad += (loss* classLabels[x] +w[i, 0]*lamda) ** 2
                    w[i, 0] -
= alpha_w * (loss* classLabels[x] * dataMatrix[x, i] + w[i, 0]*lamda)

                    for j in range(k):
                        #加入 adagrad
                        v_grad += (loss * classLabels[x]* (dataMatrix[x, i] * inter_1
[0, j] - v[i, j] * dataMatrix[x, i] * dataMatrix[x, i]) + v[i, j]*lamda) ** 2
                        v[i, j] -
= alpha_v * (loss  *classLabels[x] * (dataMatrix[x, i] * inter_1[0, j] -
 v[i, j] * dataMatrix[x, i] * dataMatrix[x, i]) + v[i, j]*lamda)
        w0_ada = np.sqrt(w0_grad)
        w_ada = np.sqrt(w_grad)
        v_ada = np.sqrt(v_grad)
        alpha_w0 = alpha/w0_ada
        alpha_w = alpha/w_ada
        alpha_v = alpha/v_ada
    return w_0, w, v
```

6、Leetcode—64. 最小路径和

```
class Solution(object):
    def minPathSum(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        m = len(grid)
        n = len(grid[0])
```

```
        dp = [[0] * n for _ in range(m)]
        dp[0][0] = grid[0][0]
        for i in range(1, m):
            dp[i][0] = grid[i][0] + dp[i-1][0]
        for j in range(1, n):
            dp[0][j] = grid[0][j] + dp[0][j-1]
        for i in range(1, m):
            for j in range(1,n):
                dp[i][j] = min(dp[i-1][j],dp[i][j-1]) + grid[i][j]
        return dp[-1][-1]
```

7、剑指 Offer 10- I. 斐波那契数列

```
class Solution:
    def fib(self, n: int) -> int:
        cur, nxt = 0, 1
        for _ in range(n):
            cur, nxt = nxt, cur + nxt
        return cur % 1000000007
```

8、Leetcode—215. 数组中的第 K 个最大元素

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        left, right, target = 0, len(nums) - 1, k -1
        while True:
            pos = self.partition(nums, left,right)
            if pos == target:
                return nums[pos]
            elif pos > target:
                right -= 1
            else:
                left += 1
    def partition(self, nums, left, right):
        tmp = nums[left]
        while left < right:
            while left < right and nums[right] <= tmp:
                right -= 1
            nums[left] = nums[right]
            while left < right and nums[left] >= tmp:
                left += 1
            nums[right] = nums[left]
        nums[left] = tmp
        return left
```

第二十篇：2022 年 4 月携程暑期实习 搜索推荐算法岗面试题 8 道

1、你所理解的推荐系统是什么样的？大致流程？所用模型？

推荐系统一方面是为了帮助消费者发现对自己有价值的商品，另一方面是帮助生产者把用户可能感兴趣的商品展现给用户，实现生产者和消费者的双赢。

大致流程主要包括：获取用户特征，召回过程，排序过程（粗排、精排）

召回模型：

• 规则召回（兴趣标签 top，热门 top，新品 top 等）

• 协同召回（基于用户的协同过滤，基于商品的协同过滤）

• 向量召回（FM 召回，Item2vec,Youtube DNN 向量召回，Graph Embedding 召回，DSSM 双塔召回）

排序模型：GBDT + LR、Xgboost、FM/FFM、Wide&Deep、DeepFM、Deep & Cross、DIN、BST 等

2、双塔模型优势，缺点，如何改进？

双塔模型的优势是速度快，但模型精度还有待提升。

速度快是因为将所有 Item 转化成 Embedding，并存储进 ANN 检索系统，比如 FAISS，以供查询。类似 FAISS 这种 ANN 检索系统对海量数据的查询效率高。

而双塔模型为了速度快，必须对用户和 Item 进行特征分离，而特征分离，又必然导致上述两个原因产生的效果损失。

改进：SENet 双塔模型，把 SENet 放在 Embedding 层之上，目的是通过 SENet 网络，动态地学习这些特征的重要性：对于每个特征学会一个特征权重，然后再把学习到的权重乘到对应特征的 Embedding 里，这样就可以动态学习特征权重，通过小权重抑制噪音或者无效低频特征，通过大权重放大重要特征影响的目的。

3、粗排的目的是什么？

粗排是用来帮精排模型找到那些它本来就要打高分的 item，只不过范围更广一些。.按照上面的例子，如果没有粗排，精排模型自己找出来的某 top10 的 item。而粗排的任务就是要找到包含这 10 个 item 的一个更小的候选集，既保证了效果，又减少线上预测的负担。

4、wide&deep 模型 为什么要有 wide 层结构，优缺点，如何改进?

wide&deep 模型中的 wide 部分可以通过利用交叉特征引入非线性高效的实现记忆能力，但需要依赖人工特征工程。

改进：DeepFM 在 Wide&Deep 的基础上进行改进，不需要预训练 FM 得到隐向量，不需要人工特征工程，能同时学习低阶和高阶的组合特征；FM 模块和 Deep 模块共享 Feature Embedding 部分，可以更快的训练，以及更精确的训练学习。

5、推荐领域 GBDT + LR 的做法了解吗?

GBDT+LR 由两部分组成，其中 GBDT 用来对训练集提取特征作为新的训练输入数据，LR 作为新训练输入数据的分类器。GBDT+LR 的提出意味着特征工程可以完全交由一个独立的模型来完成，模型的输入可以是原始的特征向量，不必在特征工程上投入过多的人工筛选和模型设计的精力，真正实现了端到端的训练。

6、粗排有哪些指标？NDCG 了解吗?

(1) NDCG，排序相似性的指标，看精排的排序结果和粗排有多相似

(2) 粗排的召回率/重叠率，粗排的 topk 和精排的 topk 有多大占比。

计算 DCG，计算公式如下：

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

其中，rel 为这个排序 list，结果 i 的一个等级得分；i 是指结果 i 的当前位置序号；

在搜索引擎中，rel 等级得分，是由人工抽样数据，并且根据一定的规则打出来的等级得分。

步骤二：

计算 IDCG（Ideal DCG），即完美序的 DCG；计算方式也同步骤 1，只是排序序列不是由算法得出，而是由人工对序列根据一定的评估准则排出来的最佳序列。

步骤三：根据前面 2 个步骤的出来的结果，计算 NDCG，计算公式如下：

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

7、ROC，PR 曲线含义，坐标轴代表什么?

ROC 曲线以真正例率 TPR 为纵轴，以假正例率 FPR 为横轴，在不同的阈值下获得坐标点，并连接各个坐标点，得到 ROC 曲线。

PR 曲线中的 P 代表的是 Precision（精准率），R 代表的是 Recall（召回率），其代表的是精准率与召回率的关系，一般情况下，Precision 设置为纵坐标，将 Recall 设置为横坐标。

8、AUC 怎么求，实际意义?

AUC：随机取一个正样本和一个负样本，正样本的预测值大于负样本预测值的概率。

AUC 计算的关键是找到所有正样本预测值大于负样本预测值的正负样本对。

首先，需要将样本按照预测值进行从小到大排序（最小 score 对应的 sample 的 rank 为 1，第二小 score 对应 sample 的 rank 为 2，以此类推）；

其次，把所有的正类样本的 rank 相加，再减去两个正样本组合的情况。

$$AUC = \frac{\sum_{i \in \text{positiveClass}} \text{rank}_i - \frac{M(1+M)}{2}}{M \times N}$$

第二十一篇：2022 年 4 月 14 日美团计算机视觉算法暑期实习面试题 6 道

1、简单介绍 gbdt 算法的原理

GBDT 是梯度提升决策树，是一种基于 Boosting 的算法，采用以决策树为基学习器的加法模型，通过不断拟合上一个弱学习器的残差，最终实现分类或回归的模型。关键在于利用损失函数的负梯度在当前模型的值作为残差的近似值，从而拟合一个回归树。对于分类问题：常使用指数损失函数；对于回归问题：常使用平方误差损失函数（此时，其负梯度就是通常意义的残差），对于一般损失函数来说就是残差的近似。

无论损失函数是什么形式，每个决策树拟合的都是负梯度。准确的说，不是用负梯度代替残差，而是当损失函数是均方损失时，负梯度刚好是残差，残差只是特例。

2、pca 属于有监督还是无监督

PCA 按有监督和无监督划分应该属于无监督学习，所以数据集有无 y 并不重要，只是改变样本 X 的属性(特征)维度。

3、介绍 svm 算法

是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机。

SVM 可分为三种：

线性可分 SVM

当训练数据线性可分时，通过最大化硬间隔（hard margin）可以学习得到一个线性分类器，即硬间隔 SVM。

线性 SVM

当训练数据不能线性可分但是近似线性可分时，通过最大化软间隔（soft margin）也可以学习到一个线性分类器，即软间隔 SVM。

非线性 SVM

当训练数据线性不可分时，通过使用核技巧（kernel trick）和最大化软间隔，可以学习到一个非线性 SVM。

SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。

SVM 如何选择核函数

Linear 核：主要用于线性可分的情形。参数少，速度快，对于一般数据，分类效果已经很理想了。

RBF 核：主要用于线性不可分的情形。参数多，分类结果非常依赖于参数。有很多人是通过训练数据的交叉验证来寻找合适的参数，不过这个过程比较耗时。

4、介绍 transformer 算法

Transformer 本身是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 个 Block，Encoder 端的 Block 包括两个模块，多头 self-attention 模块以及一个前馈神经网络模块；Decoder 端的 Block 包括三个模块，多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。

5、layernorm 和 batchnorm 的比较

Batch Normalization 是对这批样本的同一维度特征做归一化， Layer Normalization 是对这单个样本的所有维度特征做归一化。

区别：LN 中同层神经元输入拥有相同的均值和方差，不同的输入样本有不同的均值和方差；BN 中则针对不同神经元输入计算均值和方差，同一个 batch 中的输入拥有相同的均值和方差。所以，LN 不依赖于 batch 的大小和输入 sequence 的长度，因此可以用于 batchsize 为 1 和 RNN 中 sequence 的 normalize 操作。

6、Leetcode—两数之和

该题较为简单。

```
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        dic = {}
        for i, num in enumerate(nums):
            if target - num in dic:
```

```
                return [i,dic[target - num]]
            else:
                dic[num] = i
```

第二十二篇：2022 年 5 月 18 日 2023 届广联达提前批面试题 5 道

1、Transformer 中 encoder 和 decoder 的区别（结构和功能上）

结构上：Transformer 本身是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 个 Block，Encoder 端的 Block 包括两个模块，多头 self-attention 模块以及一个前馈神经网络模块；Decoder 端的 Block 包括三个模块，多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。

功能上：Transformer 中 encoder 是双向的，是编码器；decoder 是单向的，是解码器，可以用于生成任务。

2、有哪些防止过拟合的方法

数据的角度：获取和使用更多的数据（数据集增强）；

模型角度：降低模型复杂度、L1\L2\Dropout 正则化、Early stopping（提前终止）

模型融合的角度：使用 bagging 等模型融合方法。

3、L1 和 L2 正则化为什么可以防止过拟合

正规化是防止过拟合的一种重要技巧。正则化通过降低模型的复杂性， 达到避免过拟合的问题。这里的降低模型的复杂性可以理解为：

L1 将很多权重变成 0，这样起到作用的因素就会减少。

L2 使权重都趋于 0，这样就不会有某个权重占比特别大。

4、传统机器学习方法了解多少

这个可以说逻辑回归模型，逻辑回归是在数据服从伯努利分布的假设下，通过极大似然的方法，运用梯度下降法来求解参数，从而达到将数据二分类的目的。

逻辑回归的优缺点

优点：

形式简单，模型的可解释性非常好。从特征的权重可以看到不同的特征对最后结果的影响，某个特征的权重值比较高，那么这个特征最后对结果的影响会比较大。

模型效果不错。在工程上是可以接受的（作为 baseline），如果特征工程做的好，效果不会太差，并且特征工程可以并行开发，大大加快开发的速度。

训练速度较快。分类的时候，计算量仅仅只和特征的数目相关。

缺点：

准确率欠佳。因为形式非常的简单，而现实中的数据非常复杂，因此，很难达到很高的准确性。很难处理数据不平衡的问题。

5、生成式模型和判别式模型的区别并举一些例子

生成模型：学习得到联合概率分布 P(x,y)，即特征 x，共同出现的概率。

常见的生成模型：朴素贝叶斯模型，混合高斯模型，HMM 模型。

判别模型：学习得到条件概率分布 P(y|x)，即在特征 x 出现的情况下标记 y 出现的概率。

常见的判别模型：感知机，决策树，逻辑回归，SVM，CRF 等。

判别式模型：要确定一个羊是山羊还是绵羊，用判别式模型的方法是从历史数据中学习到模型，然后通过提取这只羊的特征来预测出这只羊是山羊的概率，是绵羊的概率。

生成式模型：是根据山羊的特征首先学习出一个山羊的模型，然后根据绵羊的特征学习出一个绵羊的模型，然后从这只羊中提取特征，放到山羊模型中看概率是多少，再放到绵羊模型中看概率是多少，哪个大就是哪个。

第二十三篇：2022 年 4 月大厂常考 Leetcode 面试题 6 道

1、剑指 offer 29：顺时针打印矩阵

```
class Solution(object):
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        if not matrix:
            return []
        m = len(matrix)
        n = len(matrix[0])
        up = 1
        right = n
        down = m
        left = 0
        res = [matrix[0][0]]
        i,j = 0,0
        while True:
            while j < right - 1:
                j += 1
                res.append(matrix[i][j])
            right -= 1
            if len(res) == m * n:
                return res
            while i < down -1:
                i += 1
                res.append(matrix[i][j])
            down -= 1
            if len(res) == m * n:
                return res
            while j > left:
                j -= 1
                res.append(matrix[i][j])
            left += 1
            if len(res) == m * n:
                return res
            while i > up:
                i -= 1
                res.append(matrix[i][j])
            up += 1
            if len(res) == m * n:
```

```
                return res
```

2、剑指 offer 11：旋转数组的最小数字

```
class Solution(object):
    def minArray(self, numbers):
        """
        :type numbers: List[int]
        :rtype: int
        """
        n = len(numbers)
        left = 0
        right = n - 1
        while left < right:
            mid = left + (right - left) // 2
            if numbers[mid] > numbers[right]:
                left = mid + 1
            elif numbers[mid] < numbers[right]:
                right = mid
            else:
                right -= 1
        return numbers[left]
```

3、LeetCode 69：x 的平方根

```
class Solution:
    def mySqrt(self, x: int) -> int:
        left, right = 0, x
        while left <= right:
            mid = left + (right - left) // 2
            if mid * mid <= x and (mid + 1) * (mid + 1) > x:
                return mid
            elif mid * mid > x and (mid + 1) * (mid + 1) > x:
                right = mid - 1
            else:
                left = mid + 1
```

4、LeetCode 22：括号生成

```
class Solution(object):
    def generateParenthesis(self, n):
        """
        :type n: int
```

```
        :rtype: List[str]
        """
        def backtrack(n, left, right):
            if left + right == 2 * n:
                res.append(''.join(S))
            if left < n:
                S.append('(')
                backtrack(n, left + 1, right)
                S.pop()
            if right < left:
                S.append(')')
                backtrack(n, left, right + 1)
                S.pop()
        S = []
        res = []
        backtrack(n, 0, 0)
        return res
```

5、LeetCode 123：买卖股票的最佳时机 III

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        n = len(prices)
        buy1 = buy2 = -prices[0]
        sell1 = sell2 = 0
        for i in range(1, n):
            buy1 = max(buy1, -prices[i])
            sell1 = max(sell1, buy1 + prices[i])
            buy2 = max(buy2, sell1 - prices[i])
            sell2 = max(sell2, buy2 + prices[i])
        return sell2
```

6、LeetCode 3：无重复字符的最长子串

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        start = 0
        dic = {}
        res = 0
        for end in range(len(s)):
            if s[end] in dic:
```

```
                start = max(start, dic[s[end]] + 1)
            dic[s[end]] = end
            res = max(res, end - start + 1)
        return res
```

第二十四篇：2022 年 5 月 10 日美团搜索推荐算法面试题 10 道

1、介绍下推荐系统的流程

推荐系统的流程主要包含一下几个阶段。

索引&特征: 会根据内容特性提前建立若干种类型的索引。

召回阶段: 用户请求时会从各种索引种取出千/万 条 item。

粗排阶段: 针对这上千/万条 item，进行第一遍打分，再筛选出几百条或者千条。这个阶段的排序模型一般都比较简单，能够过滤掉一些与用户兴趣明显不相关的。

精排阶段: 得到几百条 item 后，精排阶段会建立相对精细的模型，根据用户的画像，偏好，上下文，结合业务目标进行排序。一般精排后返回 50-100 条给到 engine 侧。

重排阶段: engine 侧拿到精排的 50 条 item。还会做很多的人工干预和产品逻辑，比如 item 之间的多样性，产品策略逻辑，比如热门，置顶，多种内容之间的位置混合等等。最终会返回 5-10 条左右的 item，曝光给客户端。根据业务特性，在线流程还有许多比较细的模块，比如去重服务，避免给用户推荐重复的内容。特征预处理，特征抽取等模块。

2、召回和排序的差异?

召回的目的在于减少候选的数量（尽量控制在 1000 以内），方便后续排序环节使用复杂模型精准排序；因为在短时间内评估海量候选，所以召回的关键点是个快字，受限与此与排序相比，召回的算法模型相对简单，使用的特征比较少。而排序模型相对更加复杂，更追求准确性，使用的特征也会较多。

3、结果 f1 提升的 1% 怎么保证有效性，如何保证置信呢?

实验过程中固定随机种子、多次实验取平均

4、固定随机种子后，多次实验结果相同吗?

还是会有细微差别，因为在梯度传播过程，梯度（浮点数）精度有差异，随着神经网络层数的增加，梯度差异会从小数后面的位置往前跑。只能设置浮点数精度增加来缓解这个问题。

5、召回主流的做法

主流的召回做法包括：规则召回，协同召回，基于内容语义的 I2I 召回，向量召回（基于 embedding），树召回和图召回。

6、介绍下 embedding 召回

举一个文本类 embedding 的例子。

文本类的 Embedding 可以分为两种，一种是比较传统的 word2vector、fasttext、glove 这些算法的方案，叫做词向量固定表征类算法，这些算法主要是通过分析词的出现频率来进行 Embedding 生成，不考虑文本上下文。

而另一种文本 Embedding 方法，也是目前最流行的方案是动态词表征算法，比如 Bert、ELMo、GPT，这类算法会考虑文本上下文。

7、推荐系统冷启动问题，怎么解决

1. 提供非个性化的推荐

最简单的例子就是热门排行榜，我们可以给用户推荐热门排行榜，然后等到用户数据收集到一定的时候，再切换为个性化推荐。

2. 利用用户注册信息

用户注册时提供包括用户的年龄、性别、职业、民族、学历和居住地等数据，做粗粒度的个性化。有一些网站还会让用户用文字描述他们的兴趣。

3. 利用社交网络信息

引导用户通过社交网络账号登录（需要用户授权），导入用户在社交网站上的好友信息，然后给用户推荐其好友喜欢的物品。

8、LeetCode101—对称二叉树

```
class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
```

```
            return []
        def dfs(left, right):
            if not left and not right:
                return True
            if not left or not right:
                return False
            if left.val != right.val:
                return False
            return dfs(left.left, right.right) and dfs(left.right, right.left)
        return dfs(root.left, root.right)
```

9、LeetCode3—无重复字符的最长子串

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        start = 0
        dic = {}
        res = 0
        for end in range(len(s)):
            if s[end] in dic:
                start = max(start, dic[s[end]] + 1)
            dic[s[end]] = end
            res = max(res, end - start + 1)
        return res
```

10、LeetcCode130—被围绕的区域

```
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        if not board:
            return

        n, m = len(board), len(board[0])

        def dfs(x, y):
            if not 0 <= x < n or not 0 <= y < m or board[x][y] != 'O':
                return

            board[x][y] = "A"
            dfs(x + 1, y)
            dfs(x - 1, y)
```

```
            dfs(x, y + 1)
            dfs(x, y - 1)

        for i in range(n):
            dfs(i, 0)
            dfs(i, m - 1)

        for i in range(m - 1):
            dfs(0, i)
            dfs(n - 1, i)

        for i in range(n):
            for j in range(m):
                if board[i][j] == "A":
                    board[i][j] = "O"
                elif board[i][j] == "O":
                    board[i][j] = "X"
```

第二十五篇：2022 年 4 月字节视频架构暑期实习面试题 5 道

1、图像处理的基本知识：直方图均衡化、维纳滤波、锐化的操作

直方图均衡化(Histogram Equalization)是一种增强图像对比度(Image Contrast)的方法，其主要思想是将一副图像的直方图分布通过累积分布函数变成近似均匀分布，从而增强图像的对比度。

维纳滤波器一种以最小平方为最优准则的线性滤波器。在一定的约束条件下，其输出与一给定函数（通常称为期望输出）的差的平方达到最小。

锐化滤波器则使用邻域的微分作为算子，增大邻域间像素的差值，使图像的突变部分变的更加明显。锐化的作用是加强图像的边沿和轮廓，通常也成为高通滤波器。

2、Leetcode912—排序数组

快排代码

```
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        def partition(arr, low, high):
            pivot_idx = random.randint(low, high)                    # 随机选择pivot
            arr[low], arr[pivot_idx] = arr[pivot_idx], arr[low]      # pivot放置到最左边
            pivot = arr[low]                                         # 选取最左边为pivot
            left, right = low, high     # 双指针
            while left < right:
                while left<right and arr[right] >= pivot:            # 找到右边第一个<pivot的元素
                    right -= 1
                arr[left] = arr[right]                               # 并将其移动到left处
                while left<right and arr[left] <= pivot:             # 找到左边第一个>pivot的元素
                    left += 1
                arr[right] = arr[left]                               # 并将其移动到right处
            arr[left] = pivot           # pivot放置到中间left=right处
            return left
        def quickSort(arr, low, high):
            if low >= high:             # 递归结束
                return
            mid = partition(arr, low, high)     # 以mid为分割点
```

```
            quickSort(arr, low, mid-1)          # 递归对 mid 两侧元素进行排序
            quickSort(arr, mid+1, high)
        quickSort(nums, 0, len(nums)-1)         # 调用快排函数对 nums 进行排序
        return nums
```

3、Leetcode102—二叉树层序遍历

思路：

1、如果 root 为空，直接返回 [ ]

2、定义一个数组 queue，并将 root 添加到 queue 中，再定义一个 res 数组保存结果

3、遍历 当前层 queue 的每个左子节点，右子节点，入队列，且要把 queue 更新成当前层的孩子节点列表，直到 queue 为空。

代码如下：

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        queue = [root]
        res = []
        while queue:
            res.append([node.val for node in queue])
            l1 = []
            for node in queue:
                if node.left:
                    l1.append(node.left)
                if node.right:
                    l1.append(node.right)
            queue = l1
        return res
```

4、Leetcode200—岛屿数量

深度优先搜索：对为 1 的位置进行搜索，搜索过程中需要将 1 的位置变成 0，注意边界条件。

```
class Solution(object):
    def numIslands(self, grid):
        """
        :type grid: List[List[str]]
        :rtype: int
        """
```

```
        res = 0
        nr = len(grid)
        nc = len(grid[0])
        for r in range(nr):
            for c in range(nc):
                if grid[r][c] == '1':
                    res += 1
                    self.dfs(grid, r, c)
        return res
    def dfs(self, grid, r, c):
        grid[r][c] = '0'
        nr = len(grid)
        nc = len(grid[0])
        for x, y in [(r, c-1),(r, c+1),(r-1, c),(r+1, c)]:
            if 0 <= x < nr and 0 <= y < nc and grid[x][y] == '1':
                self.dfs(grid, x, y)
```

5、Leetcode48—旋转图像

思路：先水平旋转，再对角线旋转。

```
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: None Do not return anything, modify matrix in-place instead.
        """
        # 先对角线旋转，再左右旋转
        m = len(matrix)
        n = len(matrix[0])
        for i in range(1, m):
            j = 0
            while j < i:
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
                j += 1
        for i in range(m):
            j = 0
            while j < (n // 2):
                matrix[i][j], matrix[i][n - j - 1] = matrix[i][n - j - 1] ,  matrix[i][j]
                j += 1
        return matrix
```

第二十六篇：京东健康 NLP 实习面试题 5 道

1、RNN 为什么会出现梯度消失的现象

梯度消失的原因：很难捕捉到长期的依赖关系，因为乘法梯度可以随着层的数量呈指数递减/递增。但需要注意的是 RNN 中总的梯度是不会消失的。即便梯度越传越弱，那也只是远距离的梯度消失，由于近距离的梯度不会消失，所有梯度之和便不会消失。RNN 所谓梯度消失的真正含义是，梯度被近距离梯度主导，导致模型难以学到远距离的依赖关系。

2、RNN 和 LSTM 的区别

RNN 没有细胞状态；LSTM 通过细胞状态记忆信息。

RNN 激活函数只有 tanh；LSTM 通过输入门、遗忘门、输出门引入 sigmoid 函数并结合 tanh 函数，添加求和操作，减少梯度消失和梯度爆炸的可能性。

RNN 只能够处理短期依赖问题；LSTM 既能够处理短期依赖问题，又能够处理长期依赖问题。

3、word2vec 有几种方式

word2vec 有 CBOW 模型和 Skip-Gram 模型。

CBOW 模型：输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。（Continuous Bag-of-Word）

Skip-Gram 模型：输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。

有两种改进方式：

一种是基于 Hierarchical Softmax 的，另一种是基于 Negative Sampling 的。

4、greedy search 和 beam search 的区别

greedy search 比较简单，就是贪婪式的搜索，每一步都选择概率最大的单词输出，最后组成整个句子输出。这种方法给出的结果一般情况结果比较差，因为只考虑了每一步的最优解，往往里全局最优解差距很大。

优点：计算速度快，每次取概率最大的词。 **缺点：**局部最优并不等于全局最好的，而且一旦选错了，后续生成的内容很可能也是错误的，具有错误的累加效果。

beam search 是介于全局搜索和贪婪搜索之间。使用 beam size 参数来限制在每一步保留下来的可能性词的数量。beam search 是在测试阶段为了获得更好准确性而采取的一种策略，在训练阶段无需使用。

相比暴力搜索的全局最优解，降低时间复杂度的方法就是寻找次优解，具体就是把搜索空间中的 N 减下来，每一步计算完只保留 K 个(beam size)最大的取值路径，这样时间复杂度降为 O(K*N*T)，K 取值一般比 N 小很多。这样得到的虽然不是最优解，但是在 seq2seq 模型的推理预测中可以兼顾时间和效果。

5、你了解的文本表示方法有哪些

基于 one-hot、tf-idf、textrank 等的 bag-of-words；

主题模型：LSA（SVD）、pLSA、LDA；

基于词向量的固定表征：word2vec、fastText、glove

基于词向量的动态表征：elmo、GPT、bert

对比如下：

One-hot 表示 ： 维度灾难、语义鸿沟；

矩阵分解（LSA）：利用全局语料特征，但 SVD 求解计算复杂度大；

基于 NNLM/RNNLM 的词向量：词向量为副产物，存在效率不高等问题；

word2vec、fastText：优化效率高，但是基于局部语料；

glove：基于全局预料，结合了 LSA 和 word2vec 的优点；

elmo、GPT、bert：动态特征。

第二十七篇：京东广告 NLP 实习面试题 7 道

1、Beam Search 生成的句子基本都一样，是否有方法扩展生成句子的多样性

解决方法：通过分组加入相似性惩罚，具体可以参考论文 Diverse beam search：https://arxiv.org/pdf/1610.02424.pdf。

具体方法：选择 Beam size 为 B，然后将其分为 G 组，每一组就有 B/G 个 beam，每个单独的组内跟 beam search 很像，不断延展序列，同时引入一个 dissimilarity 项来保证组与组之间有差异。

组内与 beam search 很像：从 t-1 到 t 时刻，不断的减少搜索空间（如同 beam search 一样）。

组间差异：对于 t=4 时刻，我们先对第一组输出 y（t=4），然后我们开始对第二组输出 y（t=4），但是第二组 y（t=4）的 score 不仅取决于第二组之前的 y（t=3），也取决于其与第一组的相似程度。以此类推，在 t=4 时刻对于第三组的输出，我们从上图可以看到其 score 的打分标准。这儿对于其 dissimilarity 项的计算采用的办法是 hamming diversity，这个理解起来很简单，比如这个时刻可能输出的词在上面的组出现过，我们就对这个词的分数-1，如果这个时刻可能输出的词在上面组没有出现过，我们就对这个词的分数不惩罚。

2、Layer Normalization 和 Batch Normalization 的区别，padding 对这两者有影响吗，对哪一维有影响

Batch Normalization 是对这批样本的同一维度特征做归一化， **Layer Normalization** 是对这单个样本的所有维度特征做归一化。

区别：LN 中同层神经元输入拥有相同的均值和方差，不同的输入样本有不同的均值和方差；BN 中则针对不同神经元输入计算均值和方差，同一个 batch 中的输入拥有相同的均值和方差。所以，LN 不依赖于 batch 的大小和输入 sequence 的长度，因此可以用于 batchsize 为 1 和 RNN 中 sequence 的 normalize 操作。

padding 会对 Batch Normalization 的 seq_len 这个维度有影响，计算的时候会把 padding 也算进去。

3、pytorch.Dataloader 报错出现数据维度不一样怎么解决。

在构建 dataset 重写的__getitem__方法中要返回相同长度的 tensor。

可以使用向量补全的方法来解决这个问题，把不同长度的向量补全成等长的。

4、无序数组，找 topK，要求比快排快

解题思路：堆排序，复杂度 nlogk

1）取列表前 k 个元素建立一个小根堆。堆顶就是目前第 k大的数。

2）依次向后遍历原列表，对于列表中的元素，如果小于堆顶，则忽略该元素；如果大于堆顶，则将堆顶更换为该元素，并且对堆进行一次调整；

3）遍历列表所有元素后，倒序弹出堆顶。

代码

```
#创建一个比较排序函数，供下面函数调用
def sift(li, low, high):
    i = low
    j = 2 * i + 1
    tmp = li[low]
    while j <= high:
        if j + 1 <= high and li[j+1] < li[j]:
            j = j + 1
        if li[j] < tmp:
            li[i] = li[j]
            i = j
            j = 2 * i + 1
        else:
            break
        li[i] = tmp

#解决 topk 问题的主体函数
def topk(li, k):
    heap = li[0:k]
    for i in range((k-2)//2, -1, -1):
        sift(heap, i, k-1)
    # 1.建堆
    for i in range(k, len(li)-1):
        if li[i] > heap[0]:
            heap[0] = li[i]
            sift(heap, 0, k-1)

    # 2.遍历
    for i in range(k-1, -1, -1):
        heap[0], heap[i] = heap[i], heap[0]
        sift(heap, 0, i - 1)
    # 3.出数
    return heap
```

5、Bert 里面 mask 的用处。

预训练的时候在句子编码的时候将部分词 mask，这个主要作用是用被 mask 词前后的词来去猜测 mask 掉的词是什么，因为是人为 mask 掉的，所以计算机是知道 mask 词的正确值，所以也可以判断模型猜的词是否准确。进而更好地提升 Bert 词向量的双向编码能力。

6、对于两个词怎么算他们的相似度，用基于 word embedding 的方式。

欧氏距离、曼哈顿距离、马氏距离、余弦距离、汉明距离等等。

7、Leetcode—最大子序列和

```
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n = len(nums)
        pre = 0
        res = nums[0]
        for i in range(n):
            pre = max(pre + nums[i], nums[i])
            res = max(res,pre)
        return res
```

第二十八篇：京东科技 NLP 实习面试题 10 道

1、Bert 里面为什么用 layer normalization，而不用 batch normalization，分别讲一下这两个啥意思。

Batch Normalization 是对这批样本的同一维度特征做归一化， Layer Normalization 是对这单个样本的所有维度特征做归一化。

区别：LN 中同层神经元输入拥有相同的均值和方差，不同的输入样本有不同的均值和方差；BN 中则针对不同神经元输入计算均值和方差，同一个 batch 中的输入拥有相同的均值和方差。所以，LN 不依赖于 batch 的大小和输入 sequence 的长度，因此可以用于 batchsize 为 1 和 RNN 中 sequence 的 normalize 操作。

2.、Bert 里面为什么 Q，K，V 要用三个不同的矩阵，用一个不是也行吗

如果使用相同的矩阵，相同量级的情况下，q 和 k 进行点积的值会是最大的，进行 softmax 的加权平均后，该词所占的比重会最大，使得其他词的比重很少，无法有效利用上下文信息来增强当前词的语义表示，而使用不同的 QKV 后，会很大程度减轻上述的影响。

3、Bert 和 transformer 讲一下

1 bert 只有 transformer 的 encode 结构 ，是生成语言模型

2 bert 加入了输入句子的 mask 机制，在输入的时候会随机 mask

3 模型接收两个句子作为输入，并且预测其中第二个句子是否在原始文档中也是后续句子 可以做对话机制的应答。

4 在训练 BERT 模型时，Masked LM 和 Next Sentence Prediction 是一起训练的，目标就是要最小化两种策略的组合损失函数。

4、AUC 指标讲一下

AUC：AUC 是 ROC 曲线下面的面积，AUC 可以解读为从所有正例中随机选取一个样本 A，再从所有负例中随机选取一个样本 B，分类器将 A 判为正例的概率比将 B 判为正例的概率大的可能性。AUC 反映的是分类器对样本的排序能力。AUC 越大，自然排序能力越好，即分类器将越多的正例排在负例之前。

5、Precision 和 Recall 讲一下

精确度（precision）/查准率：TP/（TP+FP）=TP/P　预测为真中，实际为正样本的概率

召回率（recall）/查全率：TP/（TP+FN）　正样本中，被识别为真的概率。

6、GBDT 和 Xgboost 的区别

1）GBDT 是机器学习算法，XGBoost 是该算法的一种工程实现

2）XGBoost 在使用 CART 作为基学习器时，加入了正则项来控制模型的复杂度，有利于防止过拟合，从而提高模型的泛化能力

3）GBDT 在模型训练时只使用了损失函数的一阶导数信息，XGBoost 对损失函数进行二阶泰勒展开，可以同时使用一阶和二阶导数

4）XGBoost 支持自定义损失函数，增强了模型的扩展性

5）传统的 GBDT 采用 CART 作为基学习器（也叫基分类器），XGBoost 支持多种类型的基学习器，包括树模型（gbtree 和 dart，dart 为一种引入 dropout 的树模型）和线性模型（gblinear），默认为 gbtree

6）传统的 GBDT 在每轮迭代时使用全部的数据，XGBoost 支持对数据进列采样，即特征采样，有利于防止过拟合，同时可以减少计算量，提高训练的效率

7）传统的 GBDT 不能支持缺失值的处理（必须填充），XGBoost 支持缺失值的处理，能够自动学习出缺失值的分裂方向（无需填充）

7、Xgboost 叶子结点的值怎么计算的

XGBoost 目标函数最终推导形式如下：

$$Obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

利用一元二次函数求最值的知识，当目标函数达到最小值 Obj*时，每个叶子结点的权重为 wj*。

具体公式如下：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad , \quad Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

每个叶子结点的权重 (score)

第 t 棵树带来的最小损失(训练损失+正则化损失)

8、LightGBM 对于 Xgboost 有什么改进

模型精度：XGBoost 和 LightGBM 相当。

训练速度：LightGBM 远快于 XGBoost。(快百倍以上，跟数据集有关系)

内存消耗：LightGBM 远小于 XGBoost。(大约是 xgb 的五分之一)

缺失值特征：XGBoost 和 LightGBM 都可以自动处理特征缺失值。

分类特征：XGBoost 不支持类别特征，需要 OneHot 编码预处理。LightGBM 直接支持类别特征。

LightGBM 在 XGBoost 上主要有 3 方面的优化。

1，Histogram 算法:直方图算法。

2，GOSS 算法:基于梯度的单边采样算法。

3，EFB 算法:互斥特征捆绑算法。

9、防止过拟合的方式

降低模型复杂度

增加更多的训练数据：使用更大的数据集训练模型

数据增强

正则化：L1、L2、添加 BN 层

添加 Dropout 策略

Early Stopping

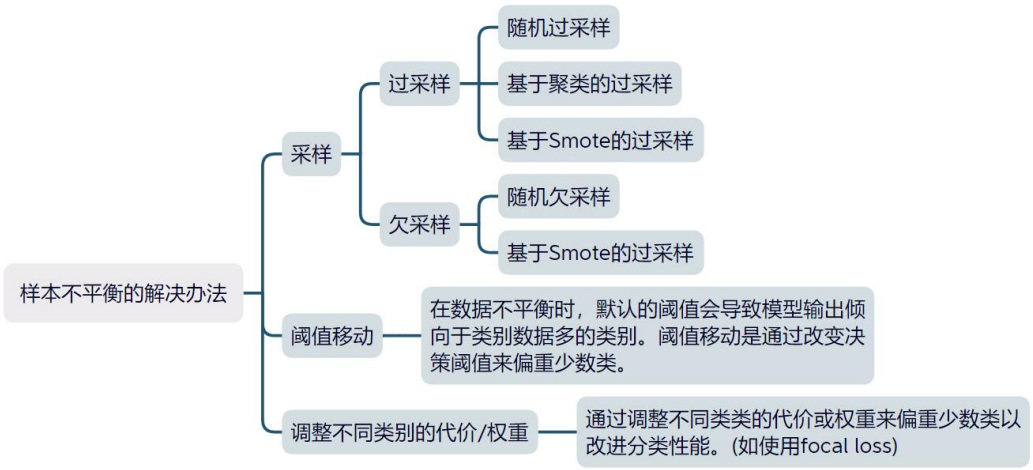
10、Adam 讲一下

Adam 算法即自适应时刻估计方法（Adaptive Moment Estimation），能计算每个参数的自适应学习率。这个方法不仅存储了 AdaDelta 先前平方梯度的指数衰减平均值，而且保持了先前梯度 M(t)的指数衰减平均值，这一点与动量类似。

Adam 实际上就是将 Momentum 和 RMSprop 集合在一起，把一阶动量和二阶动量都使用起来了。

第二十九篇：2022 年 6 月 24 日字节电商 CV 实习岗面试题 10 道

1、如何解决类别极度不平衡的问题?



2、说下 Transformer 模型

Transformer 本身是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 个 Block，Encoder 端的 Block 包括两个模块，多头 self-attention 模块以及一个前馈神经网络模块；Decoder 端的 Block 包括三个模块，多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。

3、说下 Focal Loss

Focal loss 是目标检测中解决正负样本严重不平衡的方法，在标准交叉熵损失基础上修改得到的。这个函数可以通过减少易分类样本的权重，使得模型在训练时更专注于稀疏的难分类的样本；防止大量易分类负样本在 loss 中占主导地位。

4、介绍下深度可分离卷积和传统卷积的区别

传统的卷积是各个通道上采用相 不同的卷积核，然后不同的卷积核用于提取不同方面的特征。

深度可分离卷积先在各个通道上采用不同的卷积核提取不同的特征，但是这样对于某个通道来说，就只提取了一方面的特征，因此在此基础上加入点卷积，用 1*1 的卷积对提取特征后的特征图再次提取不同方面的特征，最终产生和普通卷积相同的输出特征图。

好处是这样做大大节约了参数量。

5、如何防止过拟合

数据的角度：获取和使用更多的数据（数据集增强）；

模型角度：降低模型复杂度、L1\L2\Dropout 正则化、Early stopping（提前终止）

模型融合的角度：使用 bagging 等模型融合方法。

6、BN 在训练和测试的时候的区别？可以防止过拟合吗？

对于 BN，在训练时，是对每一批的训练数据进行归一化，也即用每一批数据的均值和方差。

而在测试时，比如进行一个样本的预测，就并没有 batch 的概念，因此，这个时候用的均值和方差是全量训练数据的均值和方差，这个可以通过移动平均法求得。

对于 BN，当一个模型训练完成之后，它的所有参数都确定了，包括均值和方差，gamma 和 bata。

BN 算法防止过拟合：在网络的训练中，BN 的使用使得一个 minibatch 中所有样本都被关联在了一起，因此网络不会从某一个训练样本中生成确定的结果，即同样一个样本的输出不再仅仅取决于样本的本身，也取决于跟这个样本同属一个 batch 的其他样本，而每次网络都是随机取 batch，这样就会使得整个网络不会朝这一个方向使劲学习。一定程度上避免了过拟合。

7、什么是 AUC?

AUC：AUC 是 ROC 曲线下面的面积，AUC 可以解读为从所有正例中随机选取一个样本 A，再从所有负例中随机选取一个样本 B，分类器将 A 判为正例的概率比将 B 判为正例的概率大的可能性。AUC 反映的是分类器对样本的排序能力。AUC 越大，自然排序能力越好，即分类器将越多的正例排在负例之前。

8、卷积核计算公式

卷积层计算公式如下：

$$O = \frac{(W - K + 2P)}{S} + 1$$

其中，W 为输入大小，K 为卷积核大小，P 为 padding 大小，S 为步幅。

如果，想保持卷积前后的特征图大小相同，通常会设定 padding 为：

$$\text{Padding} = \frac{(K - 1)}{2}$$

9、Leetcode512：数组中的第 K 大个数

两种方法：快排和堆排序

快排

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def findTopKth(low, high):
            pivot = random.randint(low, high)
            nums[low], nums[pivot] = nums[pivot], nums[low]
            base = nums[low]
            i = low
            j = low + 1
            while j <= high:
                if nums[j] > base:
                    nums[i + 1], nums[j] = nums[j], nums[i + 1]
                    i += 1
                j += 1
            nums[low], nums[i] = nums[i], nums[low]
            if i == k - 1:
                return nums[i]
            elif i > k - 1:
                return findTopKth(low, i - 1)
            else:
                return findTopKth(i + 1, high)
        return findTopKth(0, len(nums) - 1)
```

时间复杂度为：O(n)

堆排序：

```
class Solution(object):
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        heap = []
        for num in nums:
            heapq.heappush(heap, num)
            if len(heap) > k:
                heapq.heappop(heap)
        return heap[0]
```

时间复杂度为 O(nlogk)

10、Leetcode54：螺旋矩阵（二维数组的顺时针遍历）

```
class Solution(object):
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        if not matrix:
            return []
        m = len(matrix)
        n = len(matrix[0])
        up = 1
        right = n
        down = m
        left = 0
        res = [matrix[0][0]]
        i,j = 0,0
        while True:
            while j < right - 1:
                j += 1
                res.append(matrix[i][j])
            right -= 1
            if len(res) == m * n:
                return res
            while i < down -1:
                i += 1
                res.append(matrix[i][j])
            down -= 1
            if len(res) == m * n:
                return res
            while j > left:
                j -= 1
                res.append(matrix[i][j])
            left += 1
            if len(res) == m * n:
                return res
            while i > up:
                i -= 1
                res.append(matrix[i][j])
            up += 1
            if len(res) == m * n:
                return res
```