2022 最新大厂 AI 面试题

--Q1 版 109 题：含答案及解析

七月在线第 7 本内部资料

前言

不少同学已经知道，在本《2022 最新大厂 AI 面试题》之前，七月在线还整理过《名企 AI 面试 100 题》、《2021 最新大厂 AI 面试题》，都是为了帮助更多人更好的笔试面试或找工作。当然，成册背后都有着一段不平凡的历程。

2010 年 11 月，我司创始人 July 便在他的个人博客上开始整理数千道微软等公司的面试题，侧重数据结构、算法、海量数据处理，名为《微软面试 100 题系列》，截止到 2022 年 4 月底单博客内的总阅读量便已达: 18,059,140。

2017 年 9 月，July 再次和团队整理《BAT 机器学习面试 1000 题系列》，侧重各大互联网公司的机器学习、深度学习的笔试/面试题，并于 2017 年 10 月作为国内首个 AI 题库，开始在七月在线实验室公众号上连载。

2017 年 12 月，题库正式上线七月在线官网和七月在线 APP，到 2019 年 9 月，题目总数接近 4000 道。2019 年 10 月，为提高学习或复习效率，我司 AI LAB 全体同事，从七月在线官网题库页面里的 4000 道题目中精挑细选出 100 道面试题汇总成《名企 AI 面试 100 题》。

与偏经典题型的《名企 AI 面试 100 题》不同，本《2022 最新大厂 AI 面试题》侧重最新，我司 LAB 和运营部节选了今 2022 年 1 月份至 3 月底各大厂的最新面试题，包括且不限于美团、科大讯飞、荣耀、阿里、字节等，主要考察两大方面：一方面是数据结构/算法、二方面是 ML DL CV NLP 推荐等各典型/前沿模型。

然后你会发现，即便是大厂今年最新的面试题，有些题也是百面不厌的，比如

- SVM、XGBoost、CNN、RNN/LSTM、HMM、CRF、Word2Vec、Transformer、BERT 等等（如果你现在看到的是纸质版，则你可以在七月在线官网/APP 上搜相关关键词，即可找到详尽的解析）
- 各类数据结构的增删改查，比如二叉查找树、快速排序等等

本《2022 最新大厂 AI 面试题》系列会一直更新，最终目的就是帮助更多人更好的找工作，其中，题目全部来源于各面试者在各公开渠道的分享，答案和解析则由我司 LAB 整理。

当然，如果你想在更短的时间内有更大提升的话，则可以看下就业/转型为主的七月在线集训营，或在职提升为主的七月在线高级班（CV NLP 推荐），一路，与君同行。

七月在线，二零二二年四月

目录

第一篇：荣耀提前批算法 2021 年 8 月初面试题，2 道

1、简单的 N 转 K 进制输出，并判断输出是否回文。

该题主要考察进制转换

分为两步：

（1）比如求 x 的 b 进制，那么求出 x 每次对 b 取模的余数，拼接在字符串后边。这里注意如果余数大于 10 的话，需要用大写英文字母进行表示，比如'A'表示 11，'B'表示 12 等等。。。

（2）最后，将余数组成的字符串反转过来即可。

代码如下：

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 22;

int b;

// 检查是否是回文
bool check(string str)
{
    for(int i = 0 , j = str.size() - 1; i < j; i++ , j--)
    {
        if(str[i] != str[j]) return false;
    }
    return true;
}

// 进行进制转换
string get(int x)
{
    string res; // 存储 x 在 b 进制下的表示
    while(x)
    {
        int t = x % b; // 获取余数

        if(t >= 10) res += (t - 10) + 'A'; // 如果是余数是 10 以上使用 AB...来表示
        else res += t + '0'; // 否则将余数 t 添加到 res 后边，加上'0'是为了让数字 t 转化为字符 t
```

```
30.
31.         x /= b; // 记得除以b
32.     }
33.     reverse(res.begin() , res.end()); // 最后将余数倒过来
34.     return res;
35. }
36.
37. int main()
38. {
39.     cin >> b;
40.
41.     for(int i = 1; i <= 300; i++)
42.     {
43.         int t = i * i;
44.         string s = get(t); // 用b进制表示t
45.         if(check(s)) // 如果判断是回文
46.         {
47.             cout << get(i) << ' ' << s << endl;
48.         }
49.     }
50.
51.     return 0;
52. }
```

2、Leetcode-435 无重叠区间

解题思路：题目中需要移除最少的区间，使得剩下的全是无重叠区间，也就是说，需要找到最多的无重叠区间。

贪心

将列表按照区间的最大值进行排序，遍历列表，用 end 值表示当前遍历过的最大值，当当前列表的最小值大于等于 end，说明当前区间与 end 所在的区间无重叠，无重叠区间+1，end 更新为当前区间的最大值，直至遍历完成。

代码：

```
1.  class Solution:
2.      def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
3.          n = len(intervals)
4.          if n == 0: return 0
5.          ans = 1
6.          intervals.sort(key=lambda a: a[1])
7.          end = intervals[0][1]
8.          for i in range(1,n):
```

```
            if intervals[i][0] >= end:
                ans += 1
                end = intervals[i][1]
        return n - ans
```

动态规划

状态定义

dp[i]表示列表从 0 开始到位置 i 之间的最多的无重叠区间

状态转移方程

dp[i] = max(dp[i].dp[j]+1),j 是从 i 遍历到位置 0 遇到的第一个与 i 无重复区间的位置

dp[i] = max(dp[i],dp[i-1])，找到从位置 0 到位置 i 中最多的无重叠区间

代码：

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        n = len(intervals)
        if n == 0: return 0
        dp = [1] * n
        ans = 1
        intervals.sort(key=lambda a: a[1])

        for i in range(len(intervals)):
            for j in range(i - 1, -1, -1):
                if intervals[i][0] >= intervals[j][1]:
                    dp[i] = max(dp[i], dp[j] + 1)
                    break
            dp[i] = max(dp[i], dp[i - 1])
            ans = max(ans, dp[i])

        return n - ans
```

第二篇：科大讯飞提前批算法工程师面经 V 方向）2021 年 8 月，5 道

1、Pytorch 和 Tensorflow 的区别？

图创建

创建和运行计算图可能是两个框架最不同的地方。

在 pyTorch 中，图结构是动态的，这意味着图在运行时构建。

而在 TensorFlow 中，图结构是静态的，这意味着图先被“编译”然后再运行。

pyTorch 中简单的图结构更容易理解，更重要的是，还更容易调试。调试 pyTorch 代码就像调试 Python 代码一样。你可以使用 pdb 并在任何地方设置断点。调试 tensorFlow 代码可不容易。要么得从会话请求要检查的变量，要么学会使用 tensorFlow 的调试器。

灵活性

pytorch：动态计算图，数据参数在 CPU 与 GPU 之间迁移十分灵活，调试简便；

tensorflow：静态计算图，数据参数在 CPU 与 GPU 之间迁移麻烦，调试麻烦。

设备管理

pytorch：需要明确启用的设备

tensorflow：不需要手动调整，简单

2、leetcode 爬楼梯（dp,递归等方法）

递归解法：

```
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        elif n == 2:
            return 2
        else:
            s1 = self.climbStairs(n-1)
            s2 = self.climbStairs(n-2)
            return s1+s2
```

动态规划解法：

```
class Solution:
```

```
    def climbStairs(self, n: int) -> int:
        nums = [0,1,2]
        if n == 1:
            return nums[1]
        elif n == 2:
            return nums[2]
        else:
            for i in range(3,n+1):
                nums.append(nums[i-1] + nums[i-2])
        return nums[n]
```

3、torch.eval()的作用

对 BN 的影响：

对于 BN，训练时通常采用 mini-batch，所以每一批中的 mean 和 std 大致是相同的；而测试阶段往往是单个图像的输入，不存在 mini-batch 的概念。所以将 model 改为 eval 模式后，BN 的参数固定，并采用之前训练好的全局的 mean 和 std；总结就是使用全局固定的 BN。

对 dropout 的影响：

训练阶段，隐含层神经元先乘概率 P，再进行激活；而测试阶段，神经元先激活，每个隐含层神经元的输出再乘概率 P，总结来说就是顺序不同！

4、PCA 是什么？实现过程是什么，意义是什么？

主成分分析 (PCA, principal component analysis)是一种数学降维方法，利用正交变换(orthogonal transformation)把一系列可能线性相关的变量转换为一组线性不相关的新变量，也称为主成分，从而利用新变量在更小的维度下展示数据的特征。

实现过程：

一种是基于特征值分解协方差矩阵实现 PCA 算法，一种是基于 SVD 分解协方差矩阵实现 PCA 算法。

意义：

使得数据集更易使用；降低算法的计算开销；去除噪声；使得结果容易理解。

5、简述 K-means.

K-means 算法的基本思想是：以空间中 k 个点为中心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。

假设要把样本集分为 k 个类别，算法描述如下：

（1）适当选择 k 个类的初始中心，最初一般为随机选取；

（2）在每次迭代中，对任意一个样本，分别求其到 k 个中心的欧式距离，将该样本归到距离最短的中心所在的类；

（3）利用均值方法更新该 k 个类的中心的值；

（4）对于所有的 k 个聚类中心，重复（2）（3），类的中心值的移动距离满足一定条件时，则迭代结束，完成分类。

Kmeans 聚类算法原理简单，效果也依赖于 k 值和类中初始点的选择。

第三篇：2021 年 9 月，顺丰科技视觉算法工程师面试题 7 道

1、在原地即不使用任何额外的空间复杂度交换两个数。

1.相加寄存：a=a+b b=a-b a=a-b

2.位运算：b=a^b; a=a^b; b=a^b

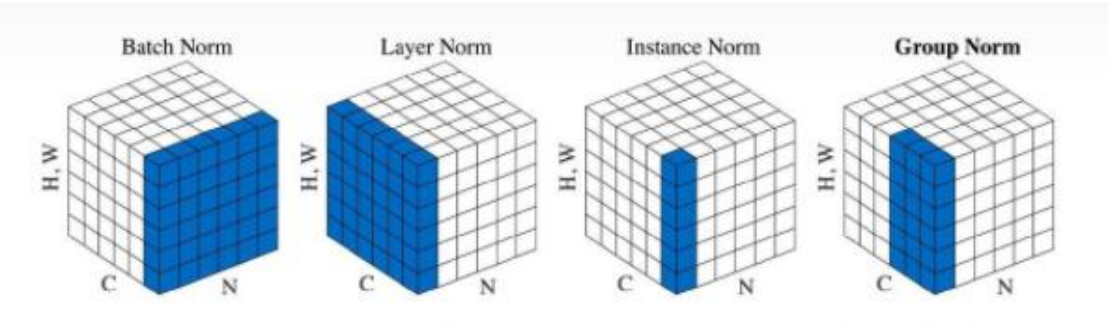
2、模型的方差和偏差是什么，怎么减少 bias 和 var。

偏差：描述的是预测值（估计值）的期望与真实值之间的差距。偏差越大，越偏离真实数据.

方差：描述的是预测值的变化范围，离散程度，也就是离其期望值的距离。方差越大，数据的分布越分散。

Baging 减少方差，boosting 减少偏差。

3、BN，GN，IN，LN 的关系。



上边这张图显示了四种标准化的方向，其中 C 轴表示通道，N 轴表示 Batch 中的样本序号，H、W 轴可以认为是图像样本中的宽和高。

Batch Normalization 会将一个 batch 中所有样本的所有 feature map 按 feature map 的序号划分为 N（N=feature map）组，然后对这 N 组内的所有像素点进行 N 次标准化；

Layer Normalization 是对一个样本中的所有数据进行标准化；

Instance Normalization 是对一个样本中的每一个通道进行单独的标准化操作；

Group Normalization 将一个样本的的通道划分为多个 group，然后在对每一个 group 进行标准化。

4、解决梯度消失的方法有哪些？

梯度修剪：grad_clip

用 ReLU、Leaky-ReLU、P-ReLU、R-ReLU、Maxout 等替代 sigmoid 函数。

用 Batch Normalization。

LSTM 的结构设计也可以改善 RNN 中的梯度消失问题。

残差网络：Resnet

5、TensorRT 的推理加速的原理是什么？

TensorRT 支持 INT8 和 FP16 的计算，通过在减少计算量和保持精度之间达到一个理想的 trade-off，达到加速推断的目的。

TensorRT 对于网络结构进行了重构和优化。主要体现在以下四个方面：

第一是 tensorRT 通过解析网络模型将网络中无用的输出层消除以减小计算。

第二是对于网络结构的垂直整合，即将目前主流神经网络的 conv、BN、Relu 三个层融合为了一个层。

第三是对于网络的水平组合，水平组合是指将输入为相同张量和执行相同操作的层融合一起。

第四是对于 concat 层，将 contact 层的输入直接送入下面的操作中，不用单独进行 concat 后在输入计算，相当于减少了一次传输吞吐。

6、介绍下 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。

7、解释一下归并排序吧。

归并排序的核心思想是分治法，将已有序的子序列合并，得到完全有序的序列。

基本过程：假设初始序列含有 n 个记录，则可以看成是 n 个有序的子序列，每个子序列的长度为 1，然后两两归并，得到 n/2 个长度为 2 或 1 的有序子序列，再两两归并，最终得到一个长度为 n 的有序序列为止，这称为 2 路归并排序。

代码示例如下：

```
1.    def merge_sort(arr):
2.      if len(arr)==1:
```

```
        return arr
    mid = len(arr)//2
    left = arr[:mid]
    right = arr[mid:]
    return marge_sort(merge_sort(left),merge_sort(right))
def marge_sort(left,right):
    result = []
    while len(left)>0 and len(right)>0:
        if left[0] < right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    result+=left
    result+=right
    return result
arr = [2,36,32,54,89,98,65,12,74]
print(merge_sort(arr))
```

第四篇：百度计算机视觉算法工程师（秋招）面试题 6 道

1、对 Transformer 的理解

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。

2、Leetcode—搜索旋转排序数组

思路一：暴力解法

直接遍历整个数组，找到目标值 target

代码如下：

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        for i,num in enumerate(nums):
            if num == target:
                return i
        return -1
```

时间复杂度：O(n)

空间复杂度：O(1)

思路二：二分查找

先要设置整个数组的左右两端端点：left = 0，right = len(nums) - 1

1、若 target == nums[mid]，直接返回

2、若 nums[left] <= nums[mid]，说明左侧区间 [left,mid]「连续递增」。此时：

若 nums[left] <= target <= nums[mid]，说明 target 位于左侧。令 right = mid-1，在左侧区间查找；否则，令 left = mid+1，在右侧区间查找

3、否则，说明右侧区间 [mid,right]「连续递增」。

此时：

若 nums[mid] <= target <= nums[right]，说明 target 位于右侧区间。令 left = mid+1，在右

侧区间查找

否则，令 right = mid-1，在左侧区间查找

代码如下：

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) -1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target:
                return mid
            elif nums[left] <= nums[mid]:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

时间复杂度：O(logn)

空间复杂度：O(1)

3、Leetcode—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、Coding：最小标记代价（类似 Leetcode 的 72 题：编辑距离）

参考：https://www.cnblogs.com/lintcode/p/14208525.html

思路：动态规划

- 已知每个整数范围[1,100]，那么对于每个元素，为了调整到该元素和与之相邻的元素的差不大于 target，该元素调整的范围就在[1,100]。所以对于数组 A[]的每一位元素，我们都需要进行[1,100]范围内的可能状态的转移。
- 令 dp[i][j]表示元素 A[i]=j 时，A[i]与 A[i-1]差值不大于 target 所需要付出的最小代价。
- 当 A[i]=j 时，可行的 A[i-1]的范围为[max(1, j - target)，min(100, j + target)]。而 dp[i][j]为所有可行的 A[i-1]中，花费代价最小的一种可能，再加上 A[i]调整到 j 所需花费 abs(j - A[i])。
- 当 A[i]=j 时，k 在[max(1, j - target)，min(100, j + target)]范围内时，我们可以写出以下式子：
- 1.临界值：
- dp[0][j] = abs(j - A[0])
- 2.状态转移方程：
- dp[i][j] = min(dp[i][j], dp[i - 1][k] + abs(j - A[i]))
- 最后在所有最后一位的可能解 dp[n-1][i]中的最小值，就是我们所求的最小代价。

代码如下：

```
public class Solution {
/*
* @param A: An integer array
* @param target: An integer
* @return: An integer
*/
public int MinAdjustmentCost(List<Integer> A, int target) {
```

```
int n = A.size();
// dp[i][j]表示元素A[i]=j 时，A[i]与A[i-1]差值不大于target 所需要付出的最小代价
int[][] dp = new int[n][101];
for (int i = 0; i < n; i++) {
for (int j = 1; j <= 100; j++) {
// 初始化为极大值
dp[i][j] = Integer.MAX_VALUE;
}
}
for (int i = 0; i < n; i++) {
for (int j = 1; j <= 100; j++) {
if (i == 0) {
// 临界值：第一个元素A[0]调整为j 的代价
dp[0][j] = Math.abs(j - A.get(0));
}
else {
// left 为A[i]=j 时，A[i-1]与A[i]差值不大于target 的A[i-1]最小值
// right 为A[i]=j 时，A[i-1]与A[i]差值不大于target 的A[i-1]最大值
int left = Math.max(1, j - target);
int right = Math.min(100, j + target);
for (int k = left; k <= right; k++) {
// 当A[i-1]=k 时，答案为A[i-1]=k 的代价dp[i-1][k]，加上A[i]=j 的调整代价abs(j-A[i])
dp[i][j] = Math.min(dp[i][j], dp[i - 1][k] + Math.abs(j - A.get(i)));
}
}

}
}
int mincost = Integer.MAX_VALUE;
for (int i = 1; i <= 100; i++) {
mincost = Math.min(mincost, dp[n - 1][i]);
}
return mincost;
}
}
```

复杂度

- 假设数组长度为 n
- 空间复杂度 O(10000*n)
- 时间复杂度 O(n^2)

5、防止过拟合的方法

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

6、数据不平衡问题处理

欠采样：从样本较多的类中再抽取，仅保留这些样本点的一部分；

过采样：复制少数类中的一些点，以增加其基数；

生成合成数据：从少数类创建新的合成点，以增加其基数。

添加额外特征：除了重采样外，我们还可以在数据集中添加一个或多个其他特征，使数据集更加丰富，这样我们可能获得更好的准确率结果。

第五篇：字节跳动计算机视觉算法工程师（秋招）面试题 3 道

1、BN 过程，为什么测试和训练不一样？

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。

2、Leeetcode：160 相交链表

方法一：暴力解法

对于 A 中的每一个结点，我们都遍历一次链表 B 查找是否存在重复结点，第一个查找到的即第一个公共结点。

```
1.  class Solution:
2.      def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
3.          p = headA
4.          while p:
5.              q = headB
6.              while q:
7.                  if p == q:
8.                      return p
9.                  q = q.next
10.             p = p.next
11.         return p
```

时间复杂度：O(n^2)

空间复杂度：O(1)

无法通过，会超时。

方法二：

对暴力解法的一个优化方案是：先将其中一个链表存到哈希表中，此时再遍历另外一个链表查找重复结点只需 O(n) 时间。

```
1.  class Solution:
2.      def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
3.          s = set()
4.          p,q = headA,headB
```

```
        while p:
            s.add(p)
            p = p.next
        while q:
            if q in s:
                return q
            q = q.next
        return None
```

时间复杂度：O(n)

空间复杂度：O(n)

方法三：走过彼此的路

利用两链表长度和相等的性质来使得两个遍历指针同步。

具体做法是：让两指针同时开始遍历，遍历到结尾的时候，跳到对方的头指针，如果有公共结点，则，会同时到达相遇的地方。

代码如下：

```
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        p,q = headA,headB
        while p != q:
            p = p.next if p else headB
            q = q.next if q else headA
        return p
```

时间复杂度：O(n)

空间复杂度：O(1)

3、概率题：两个人轮流抛硬币，抛到正面获胜，反面给对方。先抛的人胜率是多少？

先抛的人胜率是 2/3

```
A 先，B 后

P(A) = 1/2 +      //A 直接取胜
    1/2 * 1/2 * 1/2 +      // A1 失败 B1 失败 A2 取胜
    1/2 * 1/2 * 1/2 * 1/2 *1/2  +    //A1 失败 B1 失败 A2 失败 B2 失败 A3 取胜
    ...
p(A) = 1/2 + (1/2)^3 + (1/2)^5 + (1/2)^7 + ...
```

```
等比数列求和
p(A) = 1/2 * (1 - (1/4)^n) / (1 - 1/4) = 2/3

P(B） = 1/2 * 1/2 +     // A1 失败 B1 取胜
        1/2 * 1/2 * 1/2 * 1/2 +     // A1 失败 B1 失败 A2 失败 B2 取胜
        1/2 * 1/2 * 1/2 * 1/2 * 1/2 * 1/2 +  //A1 失败 B1 失败 A2 失败 B2 失败 A3 失败 B3 取胜、
        ...
p(B) = (1/2)^2 + (1/2)^4 + (1/2)^6 + ..

p(B) = 1/4 * (1 - (1/4)^n) / (1 - 1/4) = 1/3
```

第六篇：小米计算机视觉算法工程师 面试题 3 道

1、重排链表（lc143）

利用线性表存储该链表，然后利用线性表可以下标访问的特点，直接按顺序访问指定元素，重建该链表即可。

代码如下：

```
class Solution:
    def reorderList(self, head: ListNode) -> None:
        if not head:
            return

        vec = list()
        node = head
        while node:
            vec.append(node)
            node = node.next

        i, j = 0, len(vec) - 1
        while i < j:
            vec[i].next = vec[j]
            i += 1
            if i == j:
                break
            vec[j].next = vec[i]
            j -= 1

        vec[i].next = None
```

时间复杂度：O(N)

空间复杂度：O(N)

N 是链表中的节点数。

2、爬楼梯（lc70）

思路：动态规划

代码如下：

```
def climbStairs(self, n: int) -> int:
    a = b = 1
```

```
    for i in range(2, n + 1):
        a, b = b, a + b
    return b
```

3、CNN 的卷积公式

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：

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

第七篇：阿里计算机视觉算法工程师 面试题 5 道

1、手写交叉熵损失函数

二分类交叉熵

$$L=\frac{1}{N}\sum_i L_i=\frac{1}{N}\sum_i -[y_i\cdot\log(p_i)+(1-y_i)\cdot\log(1-p_i)]$$

- y_i - 表示样本 i 的label, 正类为 1 ，负类为 0
- p_i - 表示样本 i 预测为正类的概率

多分类交叉熵

$$L=\frac{1}{N}\sum_i L_i=-\frac{1}{N}\sum_i\sum_{c=1}^{M}y_{ic}\log(p_{ic})$$

其中:

- M ——类别的数量
- y_{ic} 一符号函数（0 或 1），如果样本 i 的真实类别等于 c 取 1 ，否则取 0
- p_{ic} 一观测样本 i 属于类别 c 的预测概率

2、结构风险和经验风险怎么理解

期望风险：机器学习模型关于真实分布（所有样本）的平均损失称为期望风险

经验风险：机器学习模型关于训练集的平均损失称为经验风险，当样本数无穷大∞的时候趋近于期望风险（大数定律）

结构风险：结构风险 = 经验风险 + 正则化项

经验风险是局部的，基于训练集所有样本点损失函数最小化的。

期望风险是全局的，是基于所有样本点的损失函数最小化的。

经验风险函数是现实的，可求的。

期望风险函数是理想化的，不可求的。

3、11 和 12 正则化的区别是什么，是什么原因导致的

L1/L2 的区别

L1 是模型各个参数的绝对值之和。

L2 是模型各个参数的平方和的开方值。

L1 会趋向于产生少量的特征，而其他的特征都是 0。

因为最优的参数值很大概率出现在坐标轴上，这样就会导致某一维的权重为 0 ，产生稀疏权重矩阵

L2 会选择更多的特征，这些特征都会接近于 0。

最优的参数值很小概率出现在坐标轴上，因此每一维的参数都不会是 0。当最小化||w||时，就会使每一项趋近于 0。

L1 的作用是为了矩阵稀疏化。假设的是模型的参数取值满足拉普拉斯分布。

L2 的作用是为了使模型更平滑，得到更好的泛化能力。假设的是参数是满足高斯分布。

4、BN 的计算流程和优点

BN 大致的计算流程?

1）计算样本均值。

2）计算样本方差。

3）样本数据标准化处理。

4）进行平移和缩放处理。引入了γ和β两个参数。来训练γ和β两个参数。引入了这个可学习重构参数γ、β，让我们的网络可以学习恢复出原始网络所要学习的特征分布。

BN 的优点?

1）加快训练速度，这样我们就可以使用较大的学习率来训练网络。

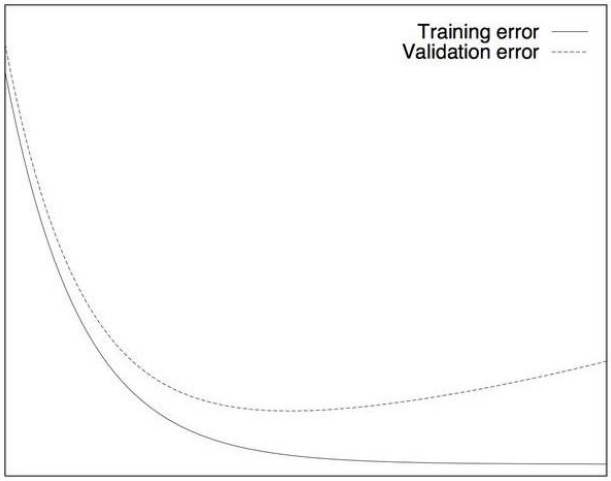
2）提高网络的泛化能力。

3）BN 层本质上是一个归一化网络层，可以替代局部响应归一化层（LRN 层）。

4）可以打乱样本训练顺序（这样就不可能出现同一张照片被多次选择用来训练）论文中提到可以提高 1%的精度。

5、模型过拟合解决方法，为什么提前停止可以解决过拟合问题？

当模型在训练集上表现很好，在验证集上表现很差的时候，则出现了过拟合情况。



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

早停：在训练中计算模型在验证集上的表现，当模型在验证集上的表现开始下降的时候，停止训练，这样就能避免继续训练导致过拟合的问题。

第八篇：美团推荐算法工程师 面试题 8 道

1、各种优化器使用的经验

梯度下降：在一个方向上更新和调整模型的参数，来最小化损失函数。

随机梯度下降（Stochastic gradient descent，SGD）对每个训练样本进行参数更新，每次执行都进行一次更新，且执行速度更快。

为了避免 SGD 和标准梯度下降中存在的问题，一个改进方法为小批量梯度下降（Mini Batch Gradient Descent），因为对每个批次中的 n 个训练样本，这种方法只执行一次更新。

使用小批量梯度下降的优点是：

1) 可以减少参数更新的波动，最终得到效果更好和更稳定的收敛。

2) 还可以使用最新的深层学习库中通用的矩阵优化方法，使计算小批量数据的梯度更加高效。

3) 通常来说，小批量样本的大小范围是从 50 到 256，可以根据实际问题而有所不同。

4) 在训练神经网络时，通常都会选择小批量梯度下降算法。

SGD 方法中的高方差振荡使得网络很难稳定收敛，所以有研究者提出了一种称为动量（Momentum）的技术，通过优化相关方向的训练和弱化无关方向的振荡，来加速 SGD 训练。

Nesterov 梯度加速法，通过使网络更新与误差函数的斜率相适应，并依次加速 SGD，也可根据每个参数的重要性来调整和更新对应参数，以执行更大或更小的更新幅度。

AdaDelta 方法是 AdaGrad 的延伸方法，它倾向于解决其学习率衰减的问题。Adadelta 不是累积所有之前的平方梯度，而是将累积之前梯度的窗口限制到某个固定大小 w。

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。

Adagrad 方法是通过参数来调整合适的学习率η，对稀疏参数进行大幅更新和对频繁参数进行小幅更新。因此，Adagrad 方法非常适合处理稀疏数据。

2、python 垃圾处理机制

在 Python 中，主要通过引用计数进行垃圾回收；通过 “标记-清除” 解决容器对象可能产生的循环引用问题；通过 “分代回收” 以空间换时间的方法提高垃圾回收效率。也就是说，python 采用的是引用计数机制为主，标记-清除和分代收集（隔代回收）两种机制为辅的策略。

3、yield 关键字作用

yield 是一个类似 return 的关键字，只是这个函数返回的是个生成器，可以节省巨大的时间、空间开销。

4、python 多继承，两个父类有同名方法怎么办？

super(Classname, self).methodname() 或 super(Classname, cls).methodname() 调用"下一个"父类中的方法

1.假设类 A 继承 B, C, D: class A(B, C, D), B/C/D 都有一个 show()方法

a.调用 B 的 show()方法:

super().show()

super(A, self).show()

b.调用 C 的 show()方法:

super(B,self).show()

c.调用 D 的 show()方法:

super(C,self).show()

2.如果在 B 类中需要调用 C 类中的 show()方法，也是一样的

```
class B:
    def show(self):
        super(B, self).show()
```

5、python 多线程/多进程/协程

类型	优点	缺点	适用
多进程 Process(multiprocessing)	可以利用 CPU 多核并行运算	占用资源最多可启动数目比线程少	CPU 密集型计算
多线程 Thread(threading)	相比进程更轻量占用资源少	相比进程，多线程只能并发执行，不能利用多 CPU（GIL）相比协程启动数目有限制，占用内存资源有线程切换开销	IO 密集型计算、同时运行的任务要求不多

多协程 Coroutine(asyncio)	内存开销最少，启动协程数量最多	支持库的限制代码实现复杂	IO 密集型计算、同时运行的较多任务

6、乐观锁/悲观锁

参考：https://zhuanlan.zhihu.com/p/82745364

悲观锁

总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁

乐观锁

总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和 CAS 算法实现。

两种锁的使用场景

乐观锁适用于写比较少的情况下（多读场景），即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。

悲观锁一般适用于多写的场景下。

两种锁的实现方式

乐观锁常见的两种实现方式

乐观锁一般会使用版本号机制或 CAS 算法实现。

悲观锁的实现方式是加锁，加锁既可以是对代码块加锁（如 Java 的 synchronized 关键字），也可以是对数据加锁（如 MySQL 中的排它锁）。

7、coding：合并两个有序链表

方法一：递归

分两种情况：空链表和非空链表

当其中一个为空链表时，可以直接返回另一个链表

当两个链表都为非空链表时，可以使用下面方法进行递归

代码如下：

```
1.  class Solution:
```

```
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        if l1 is None:
            return l2
        elif l2 is None:
            return l1
        elif l1.val < l2.val:
            l1.next = self.mergeTwoLists(l1.next,l2)
            return l1
        else:
            l2.next = self.mergeTwoLists(l1,l2.next)
            return l2
```

时间复杂度：O(m+n)

空间复杂度：O(m+n)

方法二：迭代

首先要设定一个哨兵节点，可以在最后比较容易地返回合并后的链表。

维护一个 pre 指针，根据 l1 和 l2 的大小不断更新 prev 的 next 指针。

代码如下：

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        Newhead = ListNode(-1)
        pre = Newhead
        while l1 and l2:
            if l1.val < l2.val:
                pre.next = l1
                l1 = l1.next
            else:
                pre.next = l2
                l2 = l2.next
            pre = pre.next
        if l1 is None:
            pre.next = l2
        else:
            pre.next = l1
        return Newhead.next
```

时间复杂度：O(m+n)

空间复杂度：O(1)

8、讲下 CNN 发展史

1 LeNet:

广为流传 LeNet 诞生于 1998 年，网络结构比较完整，包括卷积层、pooling 层、全连接层，这些都是现代 CNN 网络的基本组件。被认为是 CNN 的开端。

2 AlexNet:

2012 年 Geoffrey 和他学生 Alex 在 ImageNet 的竞赛中，刷新了 image classification 的记录，一举奠定了 deep learning 在计算机视觉中的地位。这次竞赛中 Alex 所用的结构就被称为作为 AlexNet。

对比 LeNet，AlexNet 加入了：

（1）非线性激活函数：ReLU；

（2）防止过拟合的方法：Dropout，Data augmentation。同时，使用多个 GPU，LRN 归一化层。其主要的优势有：网络扩大（5 个卷积层+3 个全连接层+1 个 softmax 层）；解决过拟合问题（dropout，data augmentation，LRN）；多 GPU 加速计算。

3 VGG-Net:

VGG-Net 来自 Andrew Zisserman 教授的组（Oxford），在 2014 年的 ILSVRC localization and classification 两个问题上分别取得了第一名和第二名，其不同于 AlexNet 的地方是：VGG-Net 使用更多的层，通常有 16 - 19 层，而 AlexNet 只有 8 层。同时，VGG-Net 的所有 convolutional layer 使用同样大小的 convolutional filter，大小为 3 x 3。

4 GoogLeNet:

提出的 Inception 结构是主要的创新点，这是（Network In Network）的结构，即原来的结点也是一个网络。其使用使得之后整个网络结构的宽度和深度都可扩大，能够带来 2-3 倍的性能提升。

5 Resnet

ResNet 提出了一种减轻网络训练负担的残差学习框架，这种网络比以前使用过的网络本质上层次更深。其明确地将这层作为输入层相关的学习残差函数，而不是学习未知的函数。在 ImageNet 数据集用 152 层（据说层数已经超过 1000==）——比 VGG 网络深 8 倍的深度来评估残差网络，但它仍具有较低的复杂度。在 2015 年大规模视觉识别挑战赛分类任务中赢得了第一。

第九篇：大厂常考 python 面试题 10 道

1、Python 中的列表和元组有什么区别？

list 是可变的对象，元组 tuple 是不可变的对象。也就是说列表中的元素可以进行任意修改，而元组中的元素无法修改。

2、Python 数组和列表有什么区别？

Python 中的数组和列表具有相同的存储数据方式。但是，数组只能包含单个数据类型元素，而列表可以包含任何数据类型元素。

3、Python 中 append 和 extend 的区别？

append() 向列表尾部追加一个新元素，列表只占一个索引位，在原有列表上增加

extend() 向列表尾部追加一个列表，将列表中的每个元素都追加进来，在原有列表上增加

4、Python 中==和 is 的区别

is 用于判断两个变量引用对象是否为同一个，==用于判断引用变量的值是否相等。

5、说一下 Python 深浅拷贝

对于不可变类型（字符串、数值型、布尔值）：浅拷贝和深拷贝一样，对象的引用（内存地址）没有发生变化。

对于可变对象（列表、字典、集合）：浅拷贝在拷贝时，只会 copy 一层，在内存中开辟一个空间，存放这个 copy 的列表。更深的层次并没有 copy，即第二层用的都是同一个内存；深拷贝时，会逐层进行拷贝，遇到可变类型，就开辟一块内存复制下来，遇到不可变类型就沿用之前的引用。因为不可变数据修改会从新开辟新的空间，所以，深拷贝数据之间的修改都不会相互影响。

总结如下：

浅拷贝花费时间少，占用内存少，只拷贝顶层数据，拷贝效率高。

对不可变对象拷贝时，浅拷贝和深拷贝的作用是一致的，不开辟新空间，相当于赋值操作。

可变对象浅拷贝时，只拷贝第一层中的引用，如果元素是可变对象，并且被修改，那么拷贝的对象也

会发生变化。

可变对象深拷贝时，会逐层进行拷贝，遇到可变类型，就开辟一块内存复制下来。

元组是个异类。元组是否为可变对象取决于元组中的元素。如果元组中每个元素以及其子孙元素都不包含可变对像，那么这个元祖就是不可变对象。如果元组的元素以及子孙元素中包含可变对象，那么元组就是可变对象。

6、区分下 break，continue 和 pass？

break：跳出循环，不执行下一个循环。同时 break 后面的代码也不会执行。

pass：pass 后面的代码还是会继续执行，也就是当前的循环还在继续。

continue：continue 后面的代码不会执行，而是直接进入下一个循环。

7、Python 中的局部变量和全局变量是什么？

- 全局变量：在函数外或全局空间中声明的变量称为全局变量。这些变量可以由程序中的任何函数访问。
- 局部变量：在函数内声明的任何变量都称为局部变量。此变量存在于局部空间中，而不是全局空间中。

8、python 中 range&xrange 有什么区别？

在大多数情况下，xrange 和 range 在功能方面完全相同。

它们都提供了一种生成整数列表的方法，唯一的区别是 range 返回一个 Python 列表对象，x range 返回一个 xrange 对象。这就表示 xrange 实际上在运行时并不是生成静态列表。

它使用称为 yielding 的特殊技术根据需要创建值。该技术与一种称为生成器的对象一起使用。因此如果你有一个非常巨大的列表，那么就要考虑 xrange。

9、python 装饰器是什么？

装饰器本质上是一个 Python 函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象。它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景

10、说一下 python 迭代器和生成器？

介绍 python 生成器需要先介绍可迭代对象和迭代器。

可迭代对象(Iterable Object)，简单的来理解就是可以使用 for 来循环遍历的对象。比如常见的 list、set 和 dict。

可迭代对象具有__iter__ 方法，用于返回一个迭代器，或者定义了 getitem 方法，可以按 index 索引的对象（并且能够在没有值时抛出一个 IndexError 异常），因此，可迭代对象就是能够通过它得到一个迭代器的对象。所以，可迭代对象都可以通过调用内建的 iter() 方法返回一个迭代器。

生成器其实是一种特殊的迭代器，不过这种迭代器更加优雅。它不需要再像上面的类一样写__iter__()和__next__()方法了，只需要一个 yiled 关键字。

第十篇：2022 年 2 月中旬 字节跳动算法实习生 面试题 10 道

1、LSTM 原理

LSTM 是循环神经网络 RNN 的变种，包含三个门，分别是输入门，遗忘门和输出门。

LSTM 与 GRU 区别

（1）LSTM 和 GRU 的性能在很多任务上不分伯仲；

（2）GRU 参数更少，因此更容易收敛，但是在大数据集的情况下，LSTM 性能表现更好；

（3）GRU 只有两个门（update 和 reset），LSTM 有三个门（forget，input，output），GRU 直接将 hidden state 传给下一个单元，而 LSTM 用 memory cell 把 hidden state 包装起来。

2、Transformer 的原理

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、Transformer 的计算公式，K，Q，V 怎么算

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q、K、V 分别是输入 X 线性变换得到的。

4、Transformer 为什么要用多头

多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷积核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。

5、Transformer 里的残差连接在 CV 哪里用到了（resnet），resnet 的思想

在 transformer 的 encoder 和 decoder 中，各用到了 6 层的 attention 模块，每一个 attention 模

块又和一个 FeedForward 层（简称 FFN）相接。对每一层的 attention 和 FFN，都采用了一次残差连接，即把每一个位置的输入数据和输出数据相加，使得 Transformer 能够有效训练更深的网络。在残差连接过后，再采取 Layer Nomalization 的方式。

resnet 的思想：残差模块能让训练变得更加简单，如果输入值和输出值的差值过小，那么可能梯度会过小，导致出现梯度小时的情况，残差网络的好处在于当残差为 0 时，改成神经元只是对前层进行一次线性堆叠，使得网络梯度不容易消失，性能不会下降。

6、relu 的公式，relu 在 0 的位置可导吗，不可导怎么处理

$$\mathrm{relu}(x) = \max(x, 0)$$

针对这种类型的激活函数，可以使用次梯度来解决。

次梯度方法(subgradient method)是传统的梯度下降方法的拓展，用来处理不可导的凸函数。它的优势是比传统方法处理问题范围大，劣势是算法收敛速度慢。但是，由于它对不可导函数有很好的处理方法，所以学习它还是很有必要的。

$$c <= \frac{f(x) - f(x_0)}{x - x_0}$$

对于 relu 函数，当 x>0 时，导数为 1，当 x<0 时导数为 0。因此 relu 函数在 x=0 的次梯度 c ∈ [0，1]，c 可以取[0,1]之间的任意值。

7、神经网络都有哪些正则化操作？BN 和 LN 分别用在哪？

最常用的正则化技术是 dropout，随机的丢掉一些神经元。还有数据增强，早停，L1 正则化，L2 正则化等。

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。

BN 用在图像较多，LN 用在文本较多。

8、Attention 和全连接的区别是啥？

Attention 的最终输出可以看成是一个“在关注部分权重更大的全连接层”。但是它与全连接层的区别在于，注意力机制可以利用输入的特征信息来确定哪些部分更重要。

全连接的作用的是对一个实体进行从一个特征空间到另一个特征空间的映射，而注意力机制是要对来自同一个特征空间的多个实体进行整合。全连接的权重对应的是一个实体上的每个特征的重要性，而注意

力机制的输出结果是各个实体的重要性。比如说，一个单词“love”在从 200 维的特征空间转换到 100 维的特征空间时，使用的是全连接，不需要注意力机制，因为特征空间每一维的意义是固定的。而如果我们面对的是词组“I love you”，需要对三个 200 维的实体特征进行整合，整合为一个 200 维的实体，此时就要考虑到实体间的位置可能发生变化，我们下次收到的句子可能是“love you I”，从而需要一个与位置无关的方案。

参考：https://www.zhihu.com/question/320174043/answer/651998472

9、Leetcode11. 盛水最多的容器

思路：

从两端往中间走，谁小谁动，一样的情况下谁动都行；

记录每一步 min(X, Y) * distance；

在头尾相遇时结束，也就是 O(n) 时间复杂度与 O(1) 空间复杂度。

注意：两端往中间走，移动高度小的数，每走一步计算一下。

代码：

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        l,r = 0,len(height)-1
        curMax = 0
        while l != r:
            curMax = max(curMax, min(height[l],height[r]) * (r - l))
            if height[l] < height[r]:
                l += 1
            else:
                r -= 1
        return curMax
```

10、Leetcode1884. 鸡蛋掉落-两枚鸡蛋

思路很简单，从 n 开始往下递推

比如题解中给的 n=100，从 100,99,97,94..楼分别往下丢，可以发现这些楼层的间隔是从 1,2,3,4 不断增加的，直到 34,22,9，间隔为 12,13,9（最后的间隔是因为不够了，可以看成是 14，也就是需要输出的答案）

所以直接开始推，设置 cur 为间隔，total 为总楼层，每次 total += cur + 1（+1 是因为要算上自己本身所在的楼层）

最后的间隔就是我们需要的答案了

```python
class Solution:
    def twoEggDrop(self, n: int) -> int:
        cur, total = 1, 1
        while total < n:
            total += cur + 1
            cur += 1
        return cur
```

第十一篇：大厂常考机器学习面试题 10 道

1、熵、条件熵、互信息、相对熵

熵

熵是一个随机变量不确定性的度量。对于一个离散型变量，定义为：

$$H(X) = -\sum_{x \in \chi} p(x) \log p(x)$$

一个随机性变量的熵越大，就表示不确定性越大，也就是说随机变量包含的信息量越大。

熵只依赖于 X 的分布，与 X 的取值无关。

条件熵

条件熵 H(Y|X) 表示在已知随机变量 X 的条件下随机变量 Y 的不确定性，H(Y|X) 定义为在给定条件 X 下，Y 的条件概率分布的熵对 X 的数学期望：

公式为：
$$H(Y \mid X) = \sum_{x \in X} p(x) H(Y \mid X = x)$$

互信息

互信息表示在得知 Y 后，原来信息量减少了多少。

$$I(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

如果 X 与 Y 相互独立，则互信息为 0。

KL 散度（相对熵）与 JS 散度

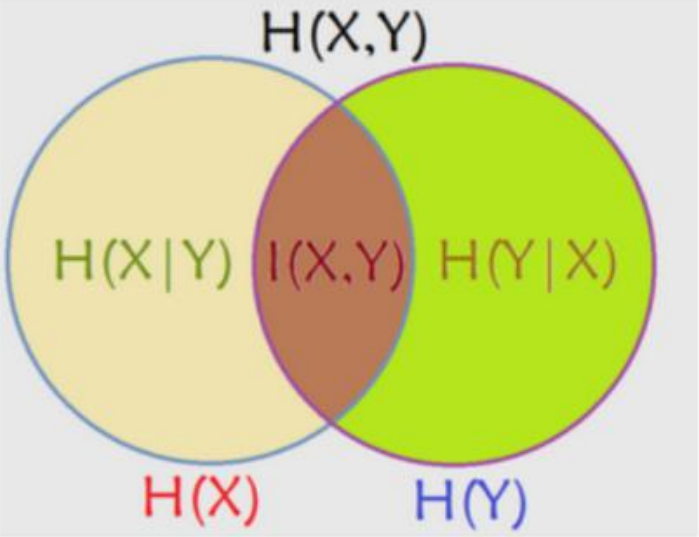
KL 散度指的是相对熵，KL 散度是两个概率分布 P 和 Q 差别的非对称性的度量。KL 散度越小表示两个分布越接近。也就是说 KL 散度是不对称的，且 KL 散度的值是非负数。

$$D_{KL}(P//Q) = -\sum_{x \in X} P(x) \log \frac{1}{P(x)} + \sum_{x \in X} P(x) \log \frac{1}{Q(x)}$$

JS 散度是基于 KL 散度的变种，度量了两个概率分布的相似度，解决了 KL 散度的非对称问题。如果两个分配 P,Q 离得很远，完全没有重叠的时候，那么 KL 散度值是没有意义的，而 JS 散度值是一个常数。这在学习算法中是比较致命的，这就意味这这一点的梯度为 0。梯度消失了。

$$JS(P_1\|P_2) = \frac{1}{2}KL\left(P_1\|\frac{P_1+P_2}{2}\right) + \frac{1}{2}KL\left(P_2\|\frac{P_1+P_2}{2}\right)$$

三者关系



2、机器学习泛化能力评测指标

泛化能力是模型对未知数据的预测能力。

准确率：分类正确的样本占总样本的比例

准确率的缺陷：当正负样本不平衡比例时，当不同类别的样本比例非常不均衡时，占比大的类别往往成为影响准确率的最主要因素。

精确率：分类正确的正样本个数占分类器预测为正样本的样本个数的比例；

召回率：分类正确的正样本个数占实际的正样本个数的比例。

F1 score：是精确率和召回率的调和平均数，综合反应模型分类的性能。

Precision 值和 Recall 值是既矛盾又统一的两个指标，为了提高 Precision 值，分类器需要尽量在“更有把握”时才把样本预测为正样本，但此时往往会因为过于保 守而漏掉很多“没有把握”的正样本，导致 Recall 值降低。

ROC 曲线的横坐标为假阳性率(False Positive Rate，FPR)；纵坐标为真阳性 率(True Positive Rate，TPR）。FPR 和 TPR 的计算方法分别为

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

假阳率（False positive rate）：FPR = FP/(FP+TN) 负样本中，被识别为真的概率

真阳率（True positive rate）：TPR = TP/（TP+FN） 正样本中，能被识别为真的概率

准确率（accuracy）：ACC =（TP+TN）/(P+N) 所有样本中，能被正确识别的概率

上式中，P 是真实的正样本的数量，N 是真实的负样本的数量，TP 是 P 个正样本中被分类器预测为正样本的个数，FP 是 N 个负样本中被分类器预测为正样本的个数。

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。

回归问题

RMSE

RMSE 经常被用来衡量回归模型的好坏。

RMSE 能够很好地反映回归模型预测值与真实值的偏离程度。但在实际问题中，如果存在个别偏离程度非常大的离群点（Outlier）时，即使离群点 数量非常少，也会让 RMSE 指标变得很差。

MAPE

引入别的评价指标，MAPE，平均绝对百分比误差

相比 RMSE，MAPE 相当于把每个点的误差进行了归一化，降低了个别离群点带来的绝对误差的影响。

F1-score

在多分类问题中，如果要计算模型的 F1-score，则有两种计算方式，分别为微观 micro-F1 和宏观 macro-F1，这两种计算方式在二分类中与 F1-score 的计算方式一样，所以在二分类问题中，计算 micro-F1=macro-F1=F1-score，micro-F1 和 macro-F1 都是多分类 F1-score 的两种计算方式。

micro-F1：

计算方法：先计算所有类别的总的 Precision 和 Recall，然后计算出来的 F1 值即为 micro-F1；

使用场景：在计算公式中考虑到了每个类别的数量，所以适用于数据分布不平衡的情况；但同时因为考虑到数据的数量，所以在数据极度不平衡的情况下，数量较多数量的类会较大的影响到 F1 的值；

marco-F1：

计算方法：将所有类别的 Precision 和 Recall 求平均，然后计算 F1 值作为 macro-F1；

使用场景：没有考虑到数据的数量，所以会平等的看待每一类（因为每一类的 precision 和 recall 都在 0-1 之间），会相对受高 precision 和高 recall 类的影响较大。

3、过拟合和欠拟合

过拟合：是指训练误差和测试误差之间的差距太大。换句换说，就是模型复杂度高于实际问题，模型在训练集上表现很好，但在测试集上却表现很差。

欠拟合：模型不能在训练集上获得足够低的误差。换句换说，就是模型复杂度低，模型在训练集上就表现很差，没法学习到数据背后的规律。

如何解决欠拟合?

欠拟合基本上都会发生在训练刚开始的时候，经过不断训练之后欠拟合应该不怎么考虑了。但是如果真的还是存在的话，可以通过增加网络复杂度或者在模型中增加特征，这些都是很好解决欠拟合的方法。

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。

4、生成式模型和判别式模型

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。

5、L1 和 L2 区别

L1 是模型各个参数的绝对值之和。

L2 是模型各个参数的平方和的开方值。

L1 会趋向于产生少量的特征，而其他的特征都是 0。 因为最优的参数值很大概率出现在坐标轴上，这样就会导致某一维的权重为 0 ，产生稀疏权重矩阵。

L2 会选择更多的特征，这些特征都会接近于 0。 最优的参数值很小概率出现在坐标轴上，因此每一维的参数都不会是 0。当最小化||w||时，就会使每一项趋近于 0。

L1 的作用是为了矩阵稀疏化。假设的是模型的参数取值满足拉普拉斯分布。

L2 的作用是为了使模型更平滑，得到更好的泛化能力。假设的是参数是满足高斯分布。

6、常见的特征选择方法

参考：https://zhuanlan.zhihu.com/p/74198735

三种：过滤法，包装法和嵌入法。

Filter：过滤法，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。

- Pearson 相关系数
- 卡方验证
- 互信息和最大信息系数
- 距离相关系数
- 方差选择法

Wrapper：包装法，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。（缺点：训练次数多，复杂度高，但效果好）

- 前向搜索：逐渐增加特征
- 后向搜索：逐渐减少特征
- 递归特征消除法：使用基模型多轮训练，每轮训练后根据得到的权值系数或者特征重要性消除系数较低的特征，再基于新的特征集进行下一轮训练。

Embedded：嵌入法，先使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。类似于 Filter 方法，但是是通过训练来确定特征的优劣。通常会使用 sklearn 中的 feature_selection 库来进行特征选择。

- 基于惩罚项的特征选择法 通过 L1 正则项来选择特征：L1 正则方法具有稀疏解的特性，因此天然具备特征选择的特性。

基于学习模型的特征排序：使用机器学习算法建立预测模型，得到打分，根据打分选择模型。

7、方差与偏差的区别

偏差：描述的是预测值（估计值）的期望与真实值之间的差距。偏差越大，越偏离真实数据.

方差：描述的是预测值的变化范围，离散程度，也就是离其期望值的距离。方差越大，数据的分布越

分散。

8、bagging、boosting、stacking 的异同

Bagging 算法(套袋发)

bagging 的算法过程如下：

从原始样本集中使用 Bootstraping 方法随机抽取 n 个训练样本，共进行 k 轮抽取，得到 k 个训练集（k 个训练集之间相互独立，元素可以有重复）。

对于 n 个训练集，我们训练 k 个模型，（这个模型可根据具体的情况而定，可以是决策树，knn 等）

对于分类问题：由投票表决产生的分类结果；对于回归问题，由 k 个模型预测结果的均值作为最后预测的结果（所有模型的重要性相同）。

Boosting（提升法）

boosting 的算法过程如下：

对于训练集中的每个样本建立权值 wi，表示对每个样本的权重， 其关键在与对于被错误分类的样本权重会在下一轮的分类中获得更大的权重（错误分类的样本的权重增加）。

同时加大分类 误差概率小的弱分类器的权值，使其在表决中起到更大的作用，减小分类误差率较大弱分类器的权值，使其在表决中起到较小的作用。每一次迭代都得到一个弱分类器，需要使用某种策略将其组合，最为最终模型，(adaboost 给每个迭代之后的弱分类器一个权值，将其线性组合作为最终的分类器,误差小的分类器权值越大。)

Bagging 和 Boosting 的主要区别

样本选择上: Bagging 采取 Bootstraping 的是随机有放回的取样，Boosting 的每一轮训练的样本是固定的，改变的是买个样的权重。

样本权重上：Bagging 采取的是均匀取样，且每个样本的权重相同，Boosting 根据错误率调整样本权重，错误率越大的样本权重会变大

预测函数上：Bagging 所以的预测函数权值相同，Boosting 中误差越小的预测函数其权值越大。

并行计算: Bagging 的各个预测函数可以并行生成;Boosting 的各个预测函数必须按照顺序迭代生成。

9、样本不平衡的解决办法

欠采样：从样本较多的类中再抽取，仅保留这些样本点的一部分；

（先对负样本做一个聚类，然后每个类中按一定的比例做采样。也可以用其他方式或者规则从负样本

中筛选区分度高的样本用于模型训练，至于哪种效果好，还是要验证的，很多时候，直接暴力一点，直接按比例负采样效果就很好。）

过采样：复制少数类中的一些点，以增加其基数；

通过数据增强扩增类别少的样本；

Focal loss：针对类别不平衡问题，用预测概率对不同类别的 loss 进行加权。Focal loss 对 CE loss 增加了一个调制系数来降低容易样本的权重值，使得训练过程更加关注困难样本。

10、余弦相似度、欧式距离与曼哈顿距离

余弦相似度用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小。

相比距离度量，余弦相似度更加注重两个向量在方向上的差异，而非距离或长度上。

欧式距离，即欧几里得距离，是最常见的两点之间的距离表示法，它定义在欧几里得空间中，

例如 x = (x1,x2,...,xn)和 y = (y1,y2,...,yn)的欧式距离可表示为：

$$d(x,y) := \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

曼哈顿距离：是欧几里得空间中两点之间的线段在坐标轴上的投影的距离的和，

例如 x = (x1,x2) y = (y1,y2)则两点的曼哈顿距离可表示为：

$$|x_1-x_2|+|y_1-y_2|$$

第十二篇：大厂常考逻辑回归模型面试题 7 道

1、简单介绍一下逻辑回归算法

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。

2、逻辑回归中参数求解方法

极大似然函数无法直接求解，一般是通过对该函数进行梯度下降来不断逼近其最优解。这里需要注意的点是要对梯度下降有一定的了解，就梯度下降本身来看的话就有随机梯度下降，批梯度下降，small batch 梯度下降三种方式，面试官可能会问这三种方式的优劣以及如何选择最合适的梯度下降方式。

批梯度下降会获得全局最优解，缺点是在更新每个参数的时候需要遍历所有的数据，计算量会很大，并且会有很多的冗余计算，导致的结果是当数据量大的时候，每个参数的更新都会很慢。

随机梯度下降是以高方差频繁更新，优点是使得 sgd 会跳到新的和潜在更好的局部最优解，缺点是使得收敛到局部最优解的过程更加的复杂。

小批量梯度下降结合了批梯度下降和随机梯度下降的优点，每次更新的时候使用 n 个样本。减少了参数更新的次数，可以达到更加稳定收敛结果，一般在深度学习当中我们采用这种方法。

3、逻辑回归中为什么使用对数损失而不用平方损失

LR 的基本表达形式如下：

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

使用交叉熵作为损失函数的梯度下降更新求导的结果如下：

首先得到损失函数如下：

$$C = \frac{1}{n}\sum [y \operatorname{In} \hat{y} + (1-y) \operatorname{In}(1-\hat{y})]$$

计算梯度如下：

$$\frac{\partial C}{\partial w} = \frac{1}{n}\sum x(\sigma(z) - y)$$

如果我们使用 MSE 作为损失函数的话，那损失函数以及求导的结果如下所示：

$$C = \frac{(y-\hat{y})^2}{2}$$

$$\frac{\partial C}{\partial w} = (\hat{y}-y)\sigma'(z)(x)$$

使用平方损失函数，会发现梯度更新的速度和 sigmod 函数本身的梯度是很相关的。sigmod 函数在它在定义域内的梯度都不大于 0.25。这样训练会非常的慢。使用交叉熵的话就不会出现这样的情况，它的导数就是一个差值，误差大的话更新的就快，误差小的话就更新的慢点，这正是我们想要的。

在使用 Sigmoid 函数作为正样本的概率时，同时将平方损失作为损失函数，这时所构造出来的损失函数是非凸的，不容易求解，容易得到其局部最优解。如果使用极大似然，其目标函数就是对数似然函数，该损失函数是关于未知参数的高阶连续可导的凸函数，便于求其全局最优解。（关于是否是凸函数，由凸函数的定义得，对于一元函数，其二阶导数总是非负，对于多元函数，其 Hessian 矩阵（Hessian 矩阵是由多元函数的二阶导数组成的方阵）的正定性来判断。如果 Hessian 矩阵是半正定矩阵，则函数是凸函数。）

4、逻辑回归公式推导

推导如下：

$$\begin{aligned}
L(w) &= \sum_i \left(y_i * \log h\left(x_i\right) + \left(1-y_i\right) * \log\left(1-h\left(x_i\right)\right)\right) \\
&= \sum_i y_i \left(\log h\left(x_i\right) - \log\left(1-h\left(x_i\right)\right)\right) + \log\left(1-h\left(x_i\right)\right) \\
&= \sum_i y_i \log \frac{h\left(x_i\right)}{1-h\left(x_i\right)} + \log\left(1-h\left(x_i\right)\right) \\
&= \sum_i y_i \left(w^T x_i\right) + \log\left(1 - \frac{1}{1+e^{-wx_i}}\right) \\
&= \sum_i \left(y_i * \left(w^T x_i\right) - \log\left(1+e^{w^T x_i}\right)\right)
\end{aligned}$$

$$\begin{aligned}
\frac{dL}{dw} &= yx - \frac{1}{1+e^{w^T x}} * e^{w^T x} * x \\
&= x(y-h(x))
\end{aligned}$$

5、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。并且逻辑回归的分布式优化 SGD 发展比较成熟。

方便调整输出结果，通过调整阈值的方式。

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。举个例子：如果我们对于一个正负样本非常不平衡的问题比如正负样本比 10000:1。我们把所有样本都预测为正也能使损失函数的值比较小。但是作为一个分类器，它对正负样本的区分能力不会很好。无法自动的进行特征筛选。只能处理二分类问题。

6、LR 和线性回归的区别

解决的任务不同：LR 主要解决的是分类问题，线性回归主要解决的是回归问题。

损失函数：线性模型是平方损失函数，而逻辑回归则是似然函数。

7、特征高度相关对逻辑回归的影响

逻辑回归在训练的过程当中，如果有很多的特征高度相关或者说有一个特征重复了很多遍，会造成怎样的影响？

如果在损失函数最终收敛的情况下，其实就算有很多特征高度相关也不会影响分类器的效果。 但是对特征本身来说的话，假设只有一个特征，在不考虑采样的情况下，你现在将它重复 N 遍。训练以后完以后，数据还是这么多，但是这个特征本身重复了 N 遍，实质上将原来的特征分成了 N 份，每一个特征都是原来特征权重值的百分之一。

为什么还是会在训练的过程当中将高度相关的特征去掉？

去掉高度相关的特征会让模型的可解释性更好；可以大大提高训练的速度。

第十三篇：大厂常考决策树模型面试题 6 道

1、ID3、C4.5、CART 树的算法思想

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。

- 经验熵 刻画了对数据集进行分类的不确定性。
- 经验条件熵 刻画了在特征 A 给定条件下，对数据集分类的不确定性。
- 信息增益 刻画了由于特征 A 的确定，从而使得对数据集的分类的不确定性减少的程度。

信息增益：数据集 D 的经验熵与关于特征 A 的经验条件熵的差值。

2、ID3、C4.5、CART 树分裂依据的公式

ID3 算法分类依据：信息增益：经验熵 - 经验条件熵

经验熵

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

经验条件熵

$$H(D\mid A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

信息增益

$$g(D,A)=H(D)-H(D\mid A)$$

C4.5 算法分类依据：信息增益比

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$$

其中，数据集 D 关于特征 A 的经验熵为

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$$

Cart 树算法分类依据：基尼指数

$$\mathrm{Gini}(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|} \right)^2$$

$$\mathrm{Gini}(D, A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \mathrm{Gini}\,(D_i)$$

3、为什么信息增益比比信息增益好？

因为信息增益会倾向于取值较多的特征，信息增益比本质上是对信息增益乘以一个加权系数，可以在一定程度上对取值较多的特征进行惩罚，避免 ID3 出现过拟合，提升决策树的泛化能力。

4、ID3、C4.5、CART 树的区别

三种树的分裂依据不同。

ID3 只能处理离散型变量；C4.5 和 CART 都可以处理连续型变量。

ID3 对缺失值比较敏感；C4.5 和 CART 都可以处理缺失值。

ID3 和 C4.5 只能用于分类任务，CART 既可以分类，也可以回归。

ID3 和 C4.5 可以在每个节点上产生多叉分支，且每个特征在层级之间不会复用，CART 每个节点只会产生两个分支，因此会形成二叉树，且每个特征可以被重复使用。

5、随机森林的大致过程和优缺点

随机森林是一种基于 bagging 的分类算法，它通过自助法（bootstrap）重采样技术，从原始训练样本集 D 中有放回地重复随机抽取 n 个样本生成新的训练样本集合训练决策树，然后按以上步骤生成 m 棵决策树组成随机森林，新数据的分类结果按分类树投票多少形成的分数而定。

随机森林大致过程如下：

1）从样本集中有放回随机采样选出 n 个样本；

2）从所有特征中随机选择 k 个特征，对选出的样本利用这些特征建立决策树（一般是 CART，也可是别的或混合）；

3）重复以上两步 m 次，即生成 m 棵决策树，形成随机森林；

4）对于新数据，经过每棵树决策，最后投票确认分到哪一类。

随机森林是一种基于 bagging 的分类算法，它通过自助法（bootstrap）重采样技术，从原始训练样本集 D 中有放回地重复随机抽取 n 个样本生成新的训练样本集合训练决策树，然后按以上步骤生成 m 棵决策树组成随机森林，新数据的分类结果按分类树投票多少形成的分数而定。

随机森林大致过程如下：

1）从样本集中有放回随机采样选出 n 个样本；

2）从所有特征中随机选择 k 个特征，对选出的样本利用这些特征建立决策树（一般是 CART，也可是别的或混合）；

3）重复以上两步 m 次，即生成 m 棵决策树，形成随机森林；

4）对于新数据，经过每棵树决策，最后投票确认分到哪一类。

优点：

每棵树都选择部分样本及部分特征，一定程度避免过拟合；

训练速度快，适合并行计算；

数据无需进行归一化

缺点：

黑盒模型，不好解释。

6、随机森林和 GBDT 区别

随机森林采用的 bagging 思想，而 GBDT 采用的 boosting 思想。这两种方法都是 Bootstrap 思想的应用，Bootstrap 是一种有放回的抽样方法思想。虽然都是有放回的抽样，但二者的区别在于：Bagging 采用有放回的均匀取样，而 Boosting 根据错误率来取样（Boosting 初始化时对每一个训练样例赋相等的权重 1 / n，然后用该算法对训练集训练 t 轮，每次训练后，对训练失败的样例赋以较大的权重），因此 Boosting 的分类精度要优于 Bagging。Bagging 的训练集的选择是随机的，各训练集之间相互独立，弱分类器可并行，而 Boosting 的训练集的选择与前一轮的学习结果有关，是串行的。

组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只能由回归树组成。

组成随机森林的树可以并行生成；而 GBDT 只能是串行生成。对于最终的输出结果而言，随机森林采用多数投票等；而 GBDT 则是将所有结果累加起来，或者加权累加起来。

随机森林对异常值不敏感；GBDT 对异常值非常敏感。

随机森林对训练集一视同仁；GBDT 是基于权值的弱分类器的集成。随机森林是通过减少模型方差提

高性能；GBDT 是通过减少模型偏差提高性能。

第十四篇：大厂常考 Xgboost 模型面试题 10 道

1、简单介绍一下 XGBoost

首先需要说一说 GBDT，它是一种基于 boosting 增强策略的加法模型，训练的时候采用前向分布算法进行贪婪的学习，每次迭代都学习一棵 CART 树来拟合之前 t-1 棵树的预测结果与训练样本真实值的残差。

XGBoost 对 GBDT 进行了一系列优化，比如损失函数进行了二阶泰勒展开、目标函数加入正则项、支持并行和默认缺失值处理等，在可扩展性和训练速度上有了巨大的提升，但其核心思想没有大的变化。

2、XGBoost 与 GBDT 有什么不同

基分类器：XGBoost 的基分类器不仅支持 CART 决策树，还支持线性分类器，此时 XGBoost 相当于带 L1 和 L2 正则化项的 Logistic 回归（分类问题）或者线性回归（回归问题）。

导数信息：XGBoost 对损失函数做了二阶泰勒展开，GBDT 只用了一阶导数信息，并且 XGBoost 还支持自定义损失函数，只要损失函数一阶、二阶可导。

正则项：XGBoost 的目标函数加了正则项， 相当于预剪枝，使得学习出来的模型更加不容易过拟合。

列抽样：XGBoost 支持列采样，与随机森林类似，用于防止过拟合。

缺失值处理：对树中的每个非叶子结点，XGBoost 可以自动学习出它的默认分裂方向。如果某个样本该特征值缺失，会将其划入默认分支。

并行化：注意不是 tree 维度的并行，而是特征维度的并行。XGBoost 预先将每个特征按特征值排好序，存储为块结构，分裂结点时可以采用多线程并行查找每个特征的最佳分割点，极大提升训练速度。

3、XGBoost 为什么使用泰勒二阶展开

精准性：相对于 GBDT 的一阶泰勒展开，XGBoost 采用二阶泰勒展开，可以更为精准的逼近真实的损失函数

可扩展性：损失函数支持自定义，只需要新的损失函数二阶可导。

4、XGBoost 防止过拟合的方法

XGBoost 在设计时，为了防止过拟合做了很多优化，具体如下：

- 目标函数添加正则项：叶子节点个数+叶子节点权重的 L2 正则化
- 列抽样：训练的时候只用一部分特征（不考虑剩余的 block 块即可）
- 子采样：每轮计算可以不使用全部样本，使算法更加保守
- shrinkage: 可以叫学习率或步长，为了给后面的训练留出更多的学习空间

5、XGBoost 如何处理缺失值

XGBoost 模型的一个优点就是允许特征存在缺失值。对缺失值的处理方式如下：

在特征 k 上寻找最佳 split point 时，不会对该列特征 missing 的样本进行遍历，而只对该列特征值为 non-missing 的样本上对应的特征值进行遍历，通过这个技巧来减少了为稀疏离散特征寻找 split point 的时间开销。

在逻辑实现上，为了保证完备性，会将该特征值 missing 的样本分别分配到左叶子结点和右叶子结点，两种情形都计算一遍后，选择分裂后增益最大的那个方向（左分支或是右分支），作为预测时特征值缺失样本的默认分支方向。

如果在训练中没有缺失值而在预测中出现缺失，那么会自动将缺失值的划分方向放到右子结点。

6、XGBoost 如何选择最佳分裂点？

XGBoost 在训练前预先将特征按照特征值进行了排序，并存储为 block 结构，以后在结点分裂时可以重复使用该结构。

因此，可以采用特征并行的方法利用多个线程分别计算每个特征的最佳分割点，根据每次分裂后产生的增益，最终选择增益最大的那个特征的特征值作为最佳分裂点。

如果在计算每个特征的最佳分割点时，对每个样本都进行遍历，计算复杂度会很大，这种全局扫描的方法并不适用大数据的场景。XGBoost 还提供了一种直方图近似算法，对特征排序后仅选择常数个候选分裂位置作为候选分裂点，极大提升了结点分裂时的计算效率。

7、XGBoost 如何评价特征的重要性

采用三种方法来评判 XGBoost 模型中特征的重要程度：

weight ： 该特征在所有树中被用作分割样本的特征的总次数。

gain ： 该特征在其出现过的所有树中产生的平均增益。

cover ： 该特征在其出现过的所有树中的平均覆盖范围。

注意：覆盖范围这里指的是一个特征用作分割点后，其影响的样本数量，即有多少样本经过该特征分割到两个子节点。

8、GBDT 与 Xgboost 的区别

- 传统的 GBDT 以 CART 树作为基学习器，XGBoost 还支持线性分类器，这个时候 XGBoost 相当于 L1 和 L2 正则化的逻辑斯蒂回归（分类）或者线性回归（回归）；
- 传统的 GBDT 在优化的时候只用到一阶导数信息，XGBoost 则对代价函数进行了二阶泰勒展开，得到一阶和二阶导数；
- XGBoost 在代价函数中加入了正则项，用于控制模型的复杂度。从权衡方差偏差来看，它降低了模型的方差，使学习出来的模型更加简单，放置过拟合，这也是 XGBoost 优于传统 GBDT 的一个特性；
- shrinkage（缩减），相当于学习速率（XGBoost 中的 eta）。XGBoost 在进行完一次迭代时，会将叶子节点的权值乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。（GBDT 也有学习速率）；
- 列抽样：XGBoost 借鉴了随机森林的做法，支持列抽样，不仅防止过 拟合，还能减少计算；
- 对缺失值的处理：对于特征的值有缺失的样本，XGBoost 还可以自动 学习出它的分裂方向；
- XGBoost 工具支持并行。Boosting 不是一种串行的结构吗?怎么并行 的？注意 XGBoost 的并行不是 tree 粒度的并行，XGBoost 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 t-1 次迭代的预测值）。XGBoost 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost 在训练之前，预先对数据进行了排序，然后保存为 block 结构，后面的迭代 中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

9、XGBoost 和 LightGBM 的区别

xgboost 采用的是 level-wise 的分裂策略，而 lightGBM 采用了 leaf-wise 的策略，区别是 xgboost 对每一层所有节点做无差别分裂，可能有些节点的增益非常小，对结果影响不大，但是 xgboost 也进行了分裂，带来了不必要的开销。 leaft-wise 的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂，如此递归进行，很明显 leaf-wise 这种做法容易过拟合，因为容易陷入比较高的深度中，因此需要对最大深度做限制，从而避免过拟合。

lightgbm 使用了基于 histogram 的决策树算法，这一点不同与 xgboost 中的 exact 算法，

histogram 算法在内存和计算代价上都有不小优势。

内存：直方图算法的内存消耗为(#data* #features * 1Bytes)(因为对特征分桶后只需保存特征离散化之后的值)，而 xgboost 的 exact 算法内存消耗为：(2 * #data * #features* 4Bytes)，因为 xgboost 既要保存原始 feature 的值，也要保存这个值的顺序索引，这些值需要 32 位的浮点数来保存。

计算：预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值，时间为(#data),而直方图算法只需要遍历桶就行了，时间为(#bin)

10、XGBoost 模型如果过拟合了怎么解决

当出现过拟合时，有两类参数可以缓解：

第一类参数：用于直接控制模型的复杂度。包括 max_depth,min_child_weight,gamma 等参数

第二类参数：用于增加随机性，从而使得模型在训练时对于噪音不敏感。包括 subsample,colsample_bytree

还有就是直接减小 learning rate，但需要同时增加 estimator 参数。

第十五篇：2022 年 3 月 30 日快手广告算法面试题 8 道

1、手写交叉熵公式

$$H_p(q) = \sum_x q(x) \log_2 \left(\frac{1}{p(x)} \right) = - \sum_x q(x) \log_2 p(x)$$

2、为什么用交叉熵不用均方误差

（1）均方误差作为损失函数，这时所构造出来的损失函数是非凸的，不容易求解，容易得到其局部最优解；而交叉熵的损失函数是凸函数；

（2）均方误差作为损失函数，求导后，梯度与 sigmoid 的导数有关，会导致训练慢；而交叉熵的损失函数求导后，梯度就是一个差值，误差大的话更新的就快，误差小的话就更新的慢点。

3、说一下 Adam 优化的优化方式

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。

4、DQN 是 On-policy 还是 off-policy？有什么区别

off-policy 的方法将收集数据作为 RL 算法中单独的一个任务，它准备两个策略：行为策略(behavior policy)与目标策略(target policy)。行为策略是专门负责学习数据的获取，具有一定的随机性，总是有一定的概率选出潜在的最优动作。而目标策略借助行为策略收集到的样本以及策略提升方法提升自身性能，并最终成为最优策略。Off-policy 是一种灵活的方式，如果能找到一个“聪明的”行为策略，总是能为算法提供最合适的样本，那么算法的效率将会得到提升。on-policy 里面只有一种策略，它既为目标策略又为行为策略。

5、value based 和 policy based 算法的区别

（1）生成 policy 上的差异：一个随机，一个确定

Value-Base 中的 action-value 估计值最终会收敛到对应的 true values（通常是不同的有限数，可

以转化为 0 到 1 之间的概率），因此通常会获得一个确定的策略（deterministic policy）

Policy-Based 不会收敛到一个确定性的值，另外他们会趋向于生成 optimal stochastic policy。如果 optimal policy 是 deterministic 的，那么 optimal action 对应的性能函数将远大于 suboptimal actions 对应的性能函数，性能函数的大小代表了概率的大小

随即策略的优点：

在很多问题中的最优策略是随机策略（stochastic policy）。（如石头剪刀布游戏，如果确定的策略对应着总出石头，随机策略对应随机出石头、剪刀或布，那么随机策略更容易获胜）

（2）一个连续，一个离散

Value-Base，对于连续动作空间问题，虽然可以将动作空间离散化处理，但离散间距的选取不易确定。过大的离散间距会导致算法取不到最优 action，会在这附近徘徊，过小的离散间距会使得 action 的维度增大，会和高维度动作空间一样导致维度灾难，影响算法的速度。

Policy-Based 适用于连续的动作空间，在连续的动作空间中，可以不用计算每个动作的概率，而是通过 Gaussian distribution （正态分布）选择 action。

（3）在 Value-Base 中，value function 的微小变化对策略的影响很大，可能直接决定了这个 action 是否被选取而 Policy-Based 避免了此缺点

6、归一化[−1, 1]和归一化到[0, 1]对后面的网络计算有什么影响吗

一般会归一化到[-1,1]，因为大部分网络是偏好零对称输入的，神经网路中使用激活函数一般都是 ReLU，如果 ReLU 的输入都是正数，那么它其实就是一个恒等函数，有没有它都一个样，ReLU 就失去了意义。如果 ReLU 的输入都是负数的话，会出现“死区”，即神经元输出都是 0，为了避免这个问题，需要令 ReLU 的输入尽量正负平衡，比如在 ReLU 前加一个 BN。

7、Leetcode16 题——三数之和

思路：排序 + 双指针

本题的难点在于如何去除重复解。

（1）判断：如果 len(nums) < 3 ，直接返回空

（2）使用 sort( )方法进行排序

（3）遍历排序后的数组

若 nums[i] > 0，后面不可能有三个数加和等于 0，直接返回结果即可。

对于重复元素，跳过，避免出现重复解。

令左指针 left = i + 1，右指针 right = n - 1，当 left < right，执行循环，三种情况：

①当满足三数之和为 0 时，需要判断左界和右界是否和下一位重复，进行去重，并更新左右指针；

②如果和大于 0，右指针左移；

③如果小于 0，左指针右移。

代码如下：

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        n = len(nums)
        res = []
        if n < 3:
            return []
        nums.sort()
        for i in range(n):
            if nums[i] > 0:
                return res
            if  i > 0 and nums[i] == nums[i -1]:
                continue
            left = i + 1
            right = n -1
            while left < right:
                if nums[i] + nums[left] + nums[right] == 0:
                    res.append([nums[i],nums[left],nums[right]])
                    while left < right and nums[left] == nums[left + 1]:
                        left += 1
                    while left < right and nums[right] == nums[right - 1]:
                        right -= 1
                    left += 1
                    right -= 1
                elif nums[i] + nums[left] + nums[right] < 0:
                    left += 1
                else:
                    right -= 1
        return res
```

时间复杂度：O(n^2)

空间复杂度：O(1)

8、Leetcode122 题——买卖股票的最佳时机

思路：动态规划

代码参考：

```
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        m = len(prices)
        dp0 = 0
        dp1 = -prices[0]
        for i in range(1, m):
            dp0 = max(dp0, dp1 + prices[i])
            dp1 = max(dp0 - prices[i], dp1)
        return dp0
```

第十六篇：2022 年 3 月 31 日美团春招推荐算法岗面试题 9 道

1、为什么分类问题损失不使用 MSE 而使用交叉熵

（1）均方误差作为损失函数，这时所构造出来的损失函数是非凸的，不容易求解，容易得到其局部最优解；而交叉熵的损失函数是凸函数；

（2）均方误差作为损失函数，求导后，梯度与 sigmoid 的导数有关，会导致训练慢；而交叉熵的损失函数求导后，梯度就是一个差值，误差大的话更新的就快，误差小的话就更新的慢点。

2、BN 的作用，除了防止梯度消失这个作用外

加快网络的训练和收敛的速度

控制梯度爆炸防止梯度消失

防止过拟合

（1）加快收敛速度：在深度神经网络中中，如果每层的数据分布都不一样的话，将会导致网络非常难收敛和训练，而如果把 每层的数据都在转换在均值为零，方差为 1 的状态下，这样每层数据的分布都是一样的训练会比较容易收敛。

（2）控制梯度爆炸防止梯度消失：以 sigmoid 函数为例，sigmoid 函数使得输出在[0,1]之间，实际上当 x 道了一定的大小，经过 sigmoid 函数后输出范围就会变得很小。

（3）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。

3、训练时出现不收敛的情况怎么办，为什么会出现不收敛

从数据角度：

是否对数据进行了预处理，包括分类标注是否正确，数据是否干净

是否对数据进行了归一化

考虑样本的信息量是否太大，而网络结构是否太简单

考虑标签是否设置正确

从模型角度：

尝试加深网络结构

Learning rate 是否合适（太大，会造成不收敛，太小，会造成收敛速度非常慢）

错误初始化网络参数

train loss 不断下降，test loss 不断下降，说明网络仍在学习;

train loss 不断下降，test loss 趋于不变，说明网络过拟合;

train loss 趋于不变，test loss 不断下降，说明数据集 100%有问题;

train loss 趋于不变，test loss 趋于不变，说明学习遇到瓶颈，需要减小学习率或批量数目;

train loss 不断上升，test loss 不断上升，说明网络结构设计不当，训练超参数设置不当，数据集经过清洗等问题。

4、LR 与决策树的区别

（1）逻辑回归通常用于分类问题，决策树可回归、可分类。

（2）逻辑回归是线性函数，决策树是非线性函数。

（3）逻辑回归的表达式很简单，回归系数就确定了模型。决策树的形式就复杂了，叶子节点的范围+取值。两个模型在使用中都有很强的解释性，银行较喜欢。

（4）逻辑回归可用于高维稀疏数据场景，比如 ctr 预估；决策树变量连续最好，类别变量的话，稀疏性不能太高。

（5）逻辑回归的核心是 sigmoid 函数，具有无限可导的优点，常作为神经网络的激活函数。

（6）在集成模型中，随机森林、GBDT 以决策树为基模型，Boosting 算法也可以用逻辑回归作为基模型。

5、有哪些决策树算法

ID3、C4.5、CART 树的算法思想

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。

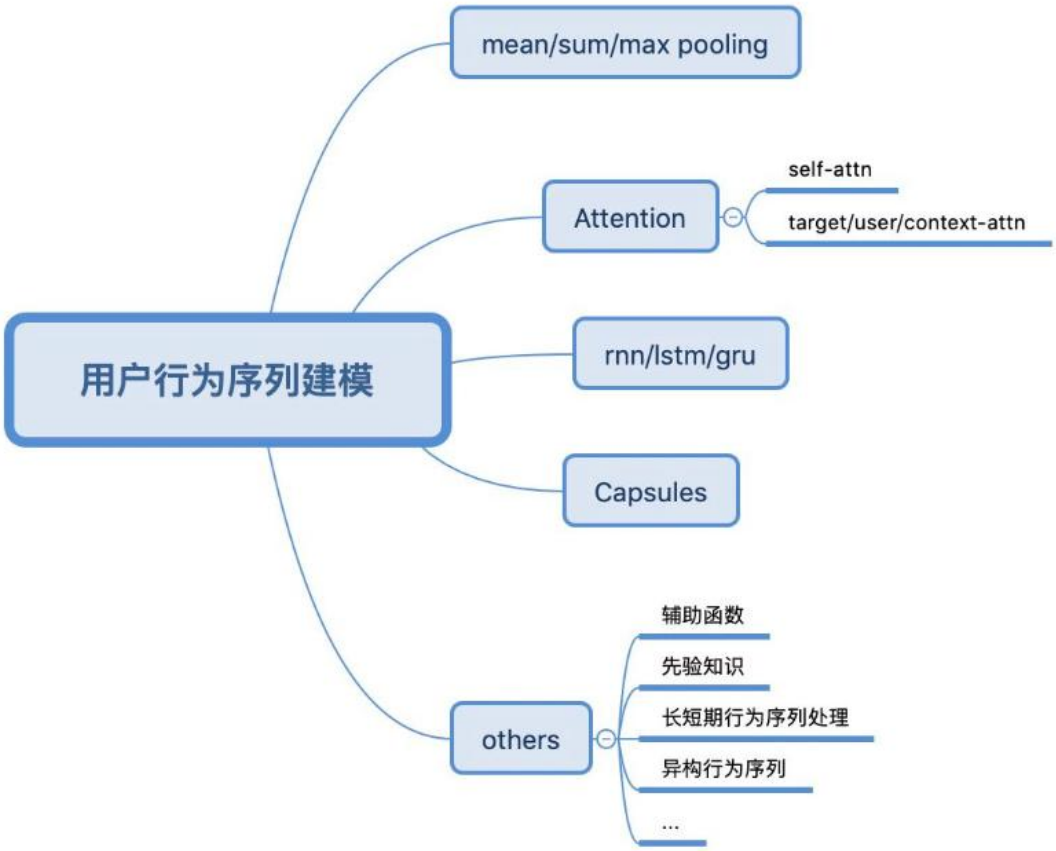
C4.5 算法的核心是在生成过程中用信息增益比来选择特征。

CART 树算法的核心是在生成过程用基尼指数来选择特征。

基于决策树的算法有随机森林、GBDT、Xgboost 等。

6、了解哪些行为序列建模方式

参考：https://zhuanlan.zhihu.com/p/138136777



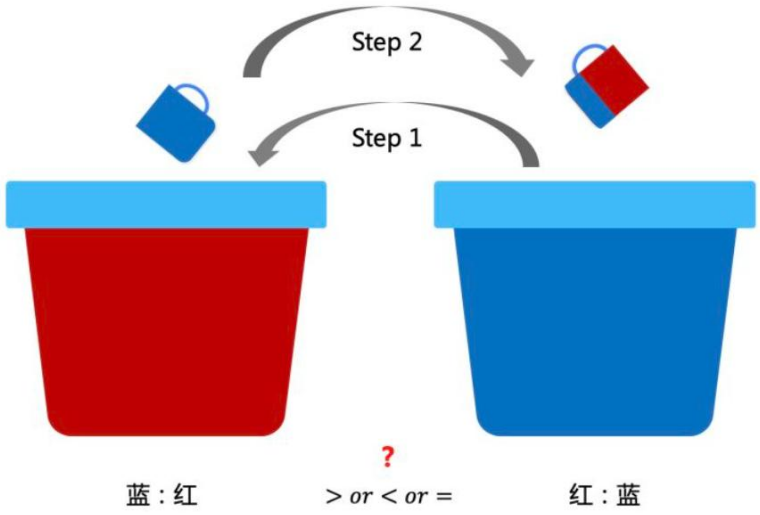
7、Leetcode_91 题：解码方法

思路：动态规划

```
class Solution:
    def numDecodings(self, s: str) -> int:
        n = len(s)
        f = [1] + [0] * n
        for i in range(1, n + 1):
            if s[i - 1] != '0':
                f[i] += f[i - 1]
            if i > 1 and s[i - 2] != '0' and int(s[i-2:i]) <= 26:
                f[i] += f[i - 2]
        return f[n]
```

8、智力题：红蓝颜料比例问题

题目描述：两个桶分别装了一样多的红色和蓝色的颜料。先从蓝色桶里舀一杯倒入红色中，搅拌不均匀。再从有蓝色的红色桶中舀一杯倒入蓝色桶里，问两个桶中蓝：红与红：蓝的大小关系？



第二步舀的时候，因为不均匀，所以无法知道具体有多少比例的红色和蓝色，可以换一个角度来考虑。因为是用的相同大小的杯子，所以两次操作后，两边的桶里的总体颜色是一样多的。假设红色里面混了一部分蓝色的颜料体积为 X 升，那么就有 X 升的红色颜料到了蓝色的桶里，所以两边的比例是一样的。

9、智力题：两个人数数，谁先数到 20 算谁赢。

要求：每次只能数 1 或者 2 个数，采取什么策略可以保证必胜，先手和后手都可以选择。

要想获胜的规律就是要抢到 19 这个数

因为是两个人参与，所以关键数是 19，当数到 19 时，对方就只能数 20 了，所以可以反推一下，要想获胜的话，在二十个数里，要想方设法地抢到 19，16，13，10，7，4，1 这几个公差为 3 的整数，也就是说在这个游戏里，要想获胜的话，就要抢到 1 这个数，即谁先数谁就获胜。