21.9.23三七面试×

一、自我介绍

二、实习

1．实习期间做的事情

2．生成模型

指针生成网络其实是PointerNetwork的延续，应用于摘要生成任务中。该网络使用generator保留了其生成能力的同时，用pointer从原文中Copy那些OOV词来保证信息正确的重复。原文更为重要的创新点是应用了coverage mechanism来解决了seq2seq的通病--repitition，这个机制可以避免在同一位置重复，也因此避免重复生成文本。

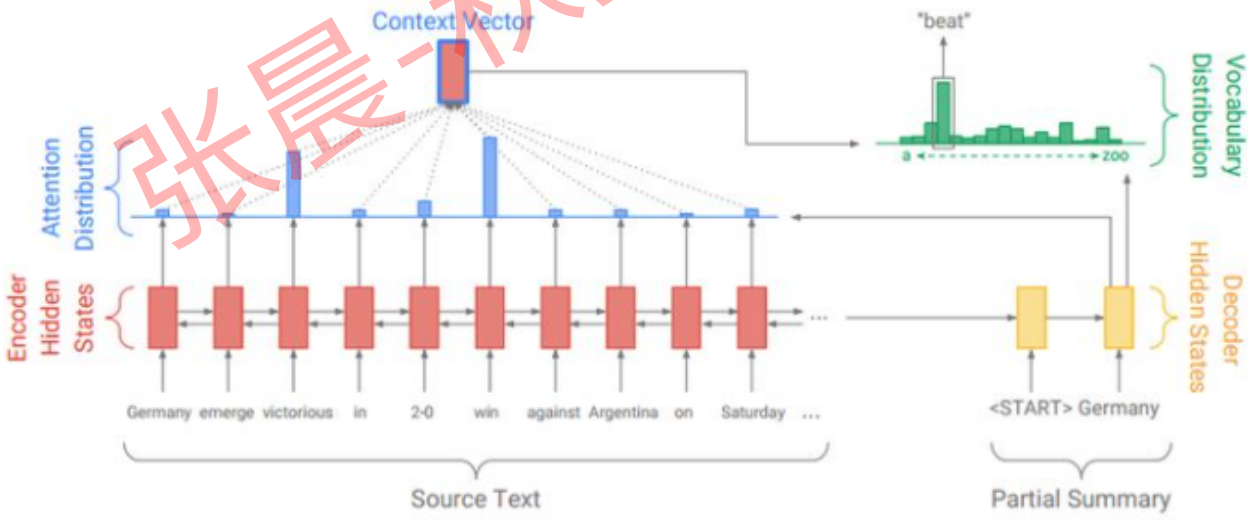


图1 Baseline Seq2Seq

正常来讲，Seq2Seq的模型结构就是两部分--编码器和解码器。正常思路是先用编码器将原文本编码成一个中间层的隐藏状态，然后用解码器来将该隐藏状态解码成为另一个文本。Baseline Seq2Seq在编码器端是一个双向的LSTM，这个双向的LSTM可以捕捉原文本的长距离依赖关系以及位置信息。编码时词嵌入经过双向LSTM后得到编码状态 h_i 。在解码器端，解码器是一个单向的LSTM，训练阶段时参考摘要词依次输入(测试阶段时是上一步的生成词)，在时间步 t 得到解码状态 s_t 。使用 h_i 和 s_t 得到该时间步原文第 i 个词注意力权重：

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{\text{attn}})$$
$$a^t = \text{softmax}(e^t)$$

得到的注意力权重和 h_i 加权求和得到重要的上下文向量 h_t^* (context vector)：

$$h_t^* = \sum_i a_i^t h_i$$

h_t^* 可以看成是该时间步通读了原文的固定尺寸的表征。然后将 s_t 和 h_t^* 经过两层线性层得到单词表分布 P_{vocab} ：

$$P_{\text{vocab}} = \text{softmax}(V'\,(V\,[s_t, h_t^*] + b) + b')$$

其中 $[s_t, h_t^*]$ 是拼接。这样就得到了一个sofmax的概率分布，就可以预测需要生成的词：

$$P(w) = P_{\text{vocab}}(w)$$

在训练阶段，时间步 t 时的损失为： $\text{loss}_t = -\log P\left(w_t^*\right)$ ，那么原输入序列的整体损失为：

$$\text{loss} = \frac{1}{T}\sum_{t=0}^{T} \text{loss}_t$$

原文中的Pointer-Generator Networks是一个混合了 Baseline seq2seq和PointerNetwork的网络，它具有Baseline seq2seq的生成能力和PointerNetwork的Copy能力。如何权衡一个词应该是生成的还是复制的？原文中引入了一个权重Pgen。

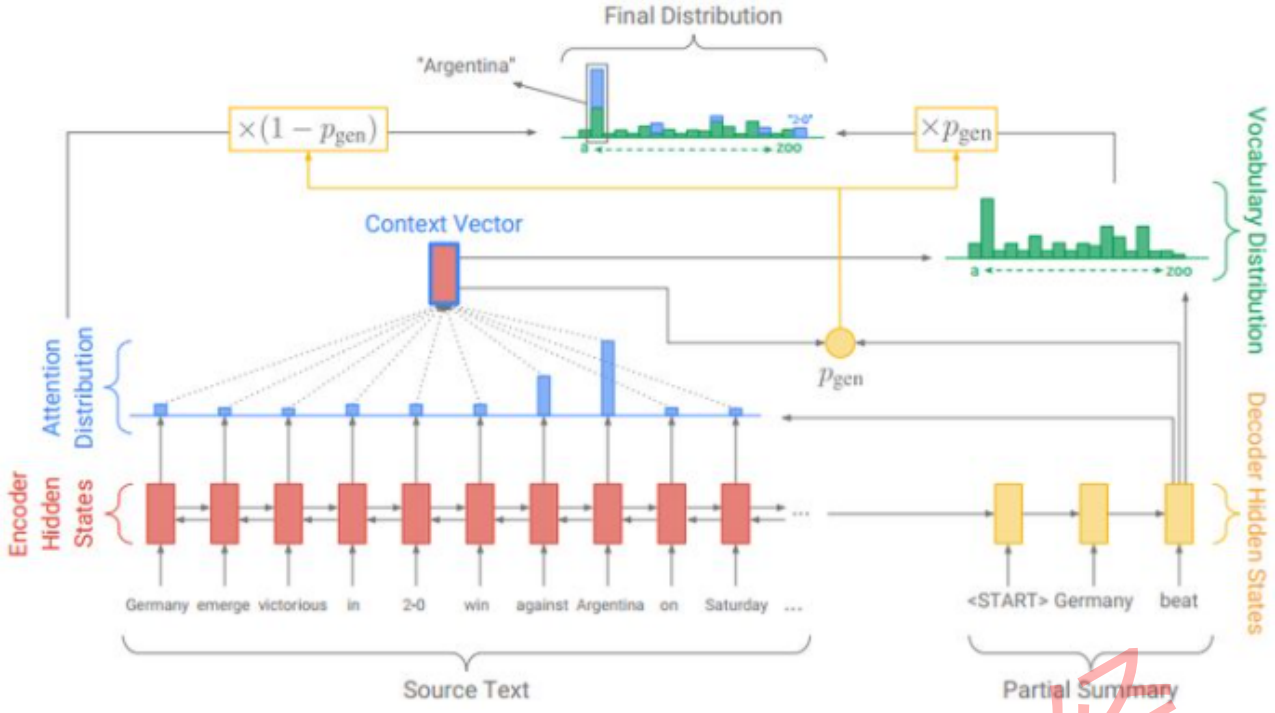


图2 Pointer-Generator Networks

从Baseline seq2seq的模型结构中得到了 s_t 和 h_t^* ，和解码器输入 x_t 一起来计算 P_{gen} ：

$$p_{\text{gen}} = \sigma\left(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{\text{ptr}}\right)$$

这时，会扩充单词表形成一个更大的单词表--扩充单词表(将原文当中的单词也加入到其中)，该时间步的预测词概率为：

$$P(w) = p_{\text{gen}} P_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} a_i^t$$

其中 a_i^t 表示的是原文档中的词。我们可以看到解码器一个词的输出概率有其是否拷贝是否生成的概率和决定。当一个词不出现在常规的单词表上时 $P_{\text{vocab}}(w)$ 为0，当该词不出现在文档中时 $\sum_{i:w_i=w} a_i^t$ 为0。

原文中最大的亮点就是运用了Coverage Mechanism来解决重复生成文本的问题：

具体实现上，就是将先前时间步的注意力权重加到一起得到所谓的覆盖向量 c^t（coverage vector），用先前的注意力权重决策来影响当前注意力权重的决策，这样就避免在同一位置重复，从而避免重复生成文本。计算上，先计算coverage vector c^t：

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

然后添加到注意力权重的计算过程中，c^t 用来计算 e_i^t：

$$e_i^t = v^T \tanh\left(W_h h_i + W_s s_t + w_c c_i^t + b_{\text{attn}}\right)$$

同时，为coverage vector添加损失是必要的，coverage loss计算方式为：

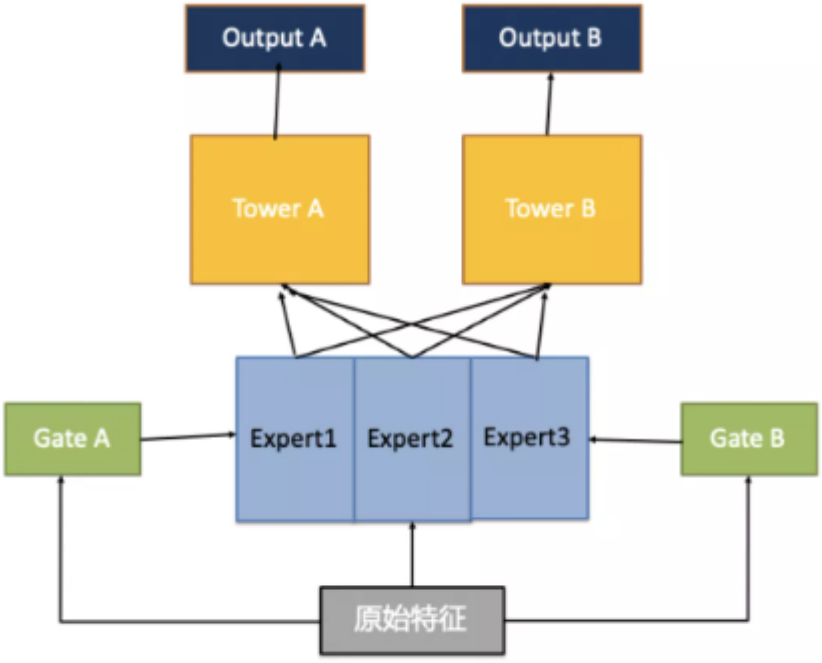
$$\text{covloss}_t = \sum_i \min\left(a_i^t, c_i^t\right)$$

这样coverage loss是一个有界的量 $covloss_t \leq \Sigma_i a_i^t = 1$ 。因此最终的LOSS为：

$$\text{loss}_t = -\log P\left(w_t^*\right) + \lambda \sum_i \min\left(a_i^t, c_i^t\right)$$

3．MMOE的原理

mmoe的思想就是，在底部的embedding层上设立多个expert网络，不同的task可以根据需求选择expert，通过expert输出的概率加权平均获得每个task的输入。每个task是单独的mlp tower。mmoe这样做可以解决什么问题呢？如果底部是完全共享的网络，对于相关性比较高的任务，效果会好一些。但是对于相关性比较差的任务，这种share-bottom的效果就不是很好。



MMoE结构

4．规则+统计的方法

四、技能

1．word2vec的两种加速方法

在CBOW和skip-gram讲解完成后，我们会发现Word2Vec模型是一个超级大的神经网络（权重矩阵规模非常大）。

举个例子，我们拥有10000个单词的词汇表，我们如果想嵌入300维的词向量，那么我们的输入-隐层权重矩阵和隐层-输出层的权重矩阵都会有 10000 x 300 = 300万个权重，在如此庞大的神经网络中进行梯度下降是相当慢的。更糟糕的是，你需要大量的训练数据来调整这些权重并且避免过拟合。百万数量级的权重矩阵和亿万数量级的训练样本意味着训练这个模型将会是个灾难。

- 根据单词出现频率构建好的huffman树，沿着路径从根节点到对应的叶子节点，一层一层的利用sigmoid函数做二分类，判断向左还是向右走，规定沿着左子树走，那么就是负类(霍夫曼树编码1)，沿着右子树走，那么就是正类(霍夫曼树编码0)。一路上的概率连乘，最终得到某个单词的输出概率。经过分层softmax优化之后，权重更新的复杂度从O（V）降至O（logV），极大提高了效率。当然，如果中心词是一个很生僻的词，还是需要在霍夫曼树中向下走很久，这也是它的一个不可避免的缺点。

- 分层softmax在每次循环迭代过程中依然要处理大量节点上的更新运算，而负采样技术只需更新“输出向量”的一部分。负抽样的目的是为了最终输出的上下文单词（正样本）[基于训练样本的半监督学习]在采样过程中应该保留下来并更新，同时也需要采集部分负样本（非上下文单词）。通过负采样，在更新隐层到输出层的权重时，只需更负采样的单词，而不用更新词汇表所有单词，从而节省巨大计算量。

2．transformer介绍一下

3．动态词向量的演变elmo gpt bert xlnet

之前那些方法构造的都是独立于上下文的word embedding，也就是无论下游任务是什么，输入的embedding始终是固定的，这就无法解决一词多义，以及在不同语境下有不同表现的需求。所以后续的ELMo，GPT以及BERT都是针对于这类问题提出的，通过预训练和fine-tune两个阶段来构造context-dependent的词表示。

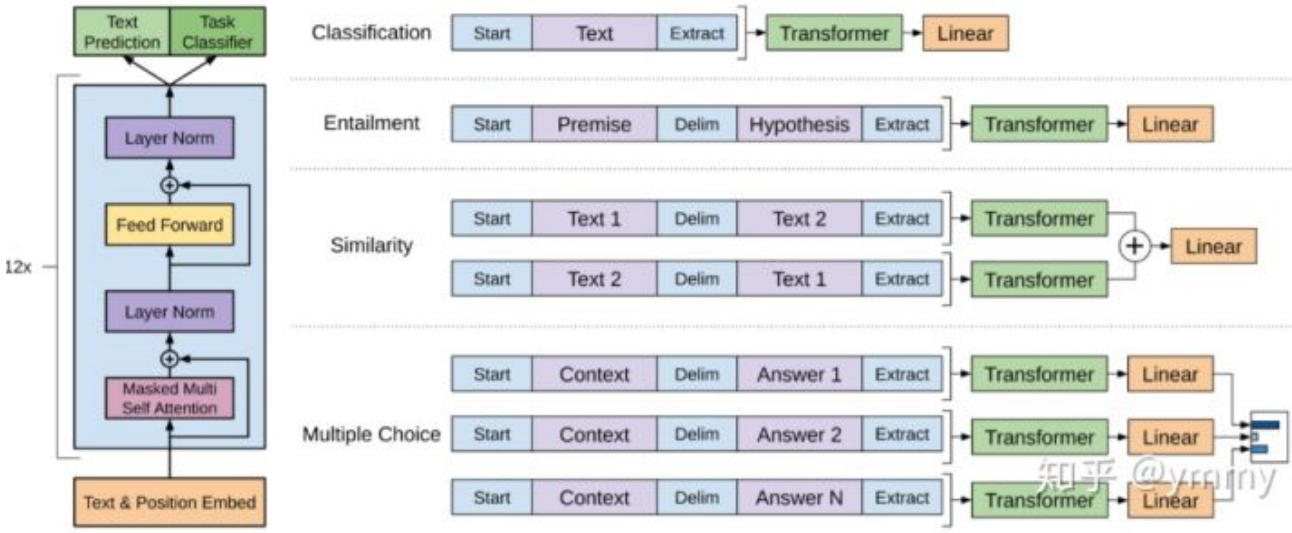
EMLo： ELMo使用双向语言模型来进行预训练，用两个分开的双层LSTM作为encoder。biLM的loss是：

$$L = \sum_{k=1}^{N} (\log p(t_k | t_1, \ldots, t_{k-1}; \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, \ldots, t_N; \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

作者认为第一层学到的是句法信息，第二层学到的是语义信息。这两层LSTM的隐状态以及初始的输入加权求和就得到当前词的embedding。ELMo还设置了一个参数，不同的下游任务可以取特定的值，来控制ELMo词向量起到的作用。

GPT： GPT和BERT与ELMo不同，ELMo使用LSTM作为编码器，而这两个用的是编码能力更强的Transformer。

GPT也是用语言模型进行大规模无监督预训练，但使用的是单向语言模型，也就是只根据上文来预测当前词。它实现的方式很直观，就是Transformer的decoder部分，只和前面的词计算self-attention来得到表示。



这是在NLP上第一次实现真正的端到端，不同的任务只需要定制不同的输入输出，无需构造内部结构。这样预训练学习到的语言学知识就能直接引入下游任务，相当于提供了先验知识。比如说人在做阅读理解时，先通读一遍全文再根据问题到文章中找回答，这些两阶段模型就类似这个过程。为了防止fine-tune时丢失预训练学到的语言知识，损失函数同时考虑下游任务loss（L_2）和语言模型loss（L_1）：

$$L_3(C) = L_2(C) + \lambda L_1(C)$$

个人认为GPT的最大创新：

- 用足够复杂的模型结构担任不同NLP任务的中间框架，启发了统一的端到端实现策略。
- 第二阶段保留语言模型的loss。

BERT：BERT被归类为自动编码器(AE)语言模型。GPT虽然效果很好，但它在预训练时使用的是transformer的decoder部分，也就是单向语言模型，在计算attention时只能看见前面的内容，这样embedding获得的上下文信息就不完整。ELMo虽然是双向语言模型，但实际上是分开执行再组合loss，这就会带来一定的损失。预训练好的Bert除了用于fine-tuning以外，还可以像ELMo一样作为特征抽取器，也就是直接用学习到的word embeddings当做其它模型的输入。

Bert最大的创新：

- 用mask策略实现了双向语言模型，非常巧妙。

- 预训练除了语言模型，还加入了next sentence prediction，试图学习更高层面的语言关联性。提供了很好的扩展思路

Bert与GPT的区别：

- GPT与Bert训练数据不同，GPT使用BooksCorpus (800M words); BERT是BooksCorpus (800M words)加Wikipedia (2,500M words)。
- GPT在预训练时没有[CLS]和[SEP]，在下游任务时才有
- GPT在fine-tuning时加入LM的loss，而Bert是完全使用任务特定的目标函数。
- GPT的lr在两阶段保持一致，Bert认为任务特定的lr效果更好

XLNet：XLNet是一种广义的自回归预训练方法。让AR语言模型从双向的上下文中学习，避免了AE语言模型中mask方法带来的弊端。AR语言模型只能使用前向或后向的上下文，如何让它学习双向上下文呢？语言模型由预训练阶段和调优阶段两个阶段组成。XLNet专注于预训练阶段。在预训练阶段，它提出了一个新的目标，称为重排列语言建模。

4．bert的两个任务的具体介绍

为了训练深度双向Transformer表示，采用了一种简单的方法：随机掩盖部分输入词，然后对那些被掩盖的词进行预测，此方法被称为“Masked LM”(MLM)。预训练的目标是构建语言模型，BERT模型采用的是bidirectional Transformer。那么为什么采用“bidirectional”的方式呢？因为在预训练语言模型来处理下游任务时，我们需要的不仅仅是某个词左侧的语言信息，还需要右侧的语言信息。文章作者在一句话中随机选择15%的词汇用于预测。对于在原句中被抹去的词汇，80%情况下采用一个特殊符号[MASK]替换，10%情况下采用一个任意词替换，剩余10%情况下保持原词汇不变。这么做的主要原因是：在后续微调任务中语句中并不会出现[MASK]标记，而且这么做的另一个好处是：预测一个词汇时，模型并不知道输入对应位置的词汇是否为正确的词汇（10%概率），这就迫使模型更多地依赖于上下文信息去预测词汇，并且赋予了模型一定的纠错能力。

很多句子级别的任务如自动问答（QA）和自然语言推理（NLI）都需要理解两个句子之间的关系，譬如上述Masked LM任务中，经过第一步的处理，15%的词汇被遮盖。那么在这一任务中我们需要随机将数据划分为等大小的两部分，一部分数据中的两个语句对是上下文连续的，另一部分数据中的两个语

句对是上下文不连续的。然后让Transformer模型来识别这些语句对中，哪些语句对是连续的，哪些对子不连续。

5．SVM的亮点与缺点

支持向量机（Support Vector Machine，常简称为SVM）是一种监督式学习的方法，可广泛地应用于统计分类以及回归分析。它是将向量映射到一个更高维的空间里，在这个空间里建立有一个最大间隔超平面。在分开数据的超平面的两边建有两个互相平行的超平面，分隔超平面使两个平行超平面的距离最大化。假定平行超平面间的距离或差距越大，分类器的总误差越小。

优点

- 有严格的数学理论支持，可解释性强，不依靠统计方法，从而简化了通常的分类和回归问题；
- 能找出对任务至关重要的关键样本（即：支持向量）；
- 采用核技巧之后，可以处理非线性分类/回归任务；
- 最终决策函数只由少数的支持向量所确定，计算的复杂性取决于支持向量的数目，而不是样本空间的维数，这在某种意义上避免了“维数灾难”。

缺点

- 当观测样本很多时，效率并不是很高；
- 对非线性问题没有通用解决方案，有时候很难找到一个合适的核函数；
- 对于核函数的高维映射解释力不强，尤其是径向基函数；
- 常规SVM只支持二分类；
- 对缺失数据敏感；

6．DT数的几种类型的区别

https://blog.csdn.net/Datawhale/article/details/102889797
<https://blog.csdn.net/Datawhale/article/details/102889797>

决策树模型在监督学习中非常常见，可用于分类（二分类、多分类）和回归。一般而言一棵“完全生长”的决策树包含，特征选择、决策树构建、剪枝三个过程。

优点：

- 决策树算法中学习简单的决策规则建立决策树模型的过程非常容易理解，

- 决策树模型可以可视化，非常直观
- 应用范围广，可用于分类和回归，而且非常容易做多类别的分类
- 能够处理数值型和连续的样本特征
- XGBoost的原理

缺点：

- 很容易在训练数据中生成复杂的树结构，造成过拟合（overfitting）。剪枝可以缓解过拟合的负作用，常用方法是限制树的高度、叶子节点中的最少样本数量。
- 学习一棵最优的决策树被认为是NP-Complete问题。实际中的决策树是基于启发式的贪心算法建立的，这种算法不能保证建立全局最优的决策树。Random Forest 引入随机能缓解这个问题

ID3缺点：

- 没有剪枝策略，容易过拟合；
- 信息增益对于可取值数目较多的特征，类似于“编号”这样的特征具有一定的偏好，因为其增益接近于1
- 只能用于处理离散分布的特征
- 没有考虑缺失值

C4.5对ID3的改进：

- 引入悲观剪枝策略
- 采用信息增益率作为特征选择的标准
- 对于连续分布的特征，先对M个值进行排序，选择相邻两个样本的平均值作为划分点，总共有M-1中划分位置，计算信息增益，选择最大的作为最终的划分点
- 对于缺失值样本，按不同的概率划分到不同的节点中

剪枝策略：

从策略上来讲，剪枝可以分为两种：预剪枝（Pre-Pruning）和后剪枝（Post-Pruning）。预剪枝是在构建决策树的时候同时进行剪枝工作，当发现分类有偏差时就及早停止（比如决定在某个节点不再分裂或划分训练元组的子集），一旦停止，该节点就成为叶子节点。

预剪枝的方法有很多，如

- 提前设定决策树的高度，当达到这个高度时，就停止构建决策树；

- 当达到某节点的实例具有相同的特征向量，也可以停止树的生长；
- 提前设定某个阈值，当达到某个节点的样例个数小于该阈值的时候便可以停止树的生长，但这种方法的缺点是对数据量的要求较大，无法处理数据量较小的训练样例；
- 同样是设定某个阈值，每次扩展决策树后都计算其对系统性能的增益，若小于该阈值，则让它停止生长。

预剪枝的显著缺点是视野效果。因为剪枝是伴随着构建决策树同时进行的，构建者无法预知下一步可能会发生的事，会出现某种情况，在相同标准下，当前决策树不满足要求最开始的构建要求、构建者进行了剪枝，但实际上若进行进一步构建后、决策树又满足了要求。这种情况下，预剪枝会过早停止决策树的生长。

后剪枝是人们普遍关注的决策树剪枝策略，与预剪枝恰好相反，后剪枝的执行步骤是先构造完成完整的决策树，再通过某些条件遍历树进行剪枝，其主要思路是通过删除节点的分支并用树叶节点替换，剪去完全成长的树的子树，这个节点所标识的类别通过大多数原则（majority class criterion）确定（既将一些子树删除而用叶子节点代替，这个叶节点所表示的类别用这棵子树中的大多数训练样本所属类别来进行标识），其类别称为Majority Class。

目前主要应用的后剪枝方法有四种：悲观错误剪枝（Pessimistic Error Pruning，PEP），最小错误剪枝（Minimum Error Pruning，MEP），代价复杂度剪枝（Cost-Complexity Pruning，CCP），基于错误的剪枝（Error-Based Pruning，EBP）。C4.5算法采用悲观错误剪枝，CART采用代价复杂度剪枝。

缺失值处理方法：

第一步，计算所有特征的信息增益或者信息增益率的时候，假设数据集一共10000个样本，特征A中缺失了5000个，则无视缺失值，在剩下的5000个特征中计算信息增益（或者信息增益率），最后乘以0.5，思想就是缺失值多的特征通过这种降低权重的方式来体现信息的缺失；

第二步，如果运气不好，正好这个A特征乘0.5之后得到的信息增益或者增益率还是最大的，那么就像西瓜书中提到的那样，存在缺失值的样板按照比例进入分裂之后的新的分支，假设根据特征A分裂得到两个新的分支，一个分支

有2000个样本，一个分支有3000个样本，则按照比例2000个缺失值和3000个缺失值样本分别进入两个分支。

C4.5的缺点：

- C4.5使用的是多叉树，效率比较慢
- C4.5只能用于分类任务，无法用于解决回归问题
- 计算公式中有log，计算复杂耗时
- 对于连续的特征，需要对值进行排序，当数据量很大时，浪费内存

CART树对C4.5的改进：

- 使用二叉树
- 适用于分类和回归任务
- 用基尼系数代替信息增益，减少log计算
- 采用代理分裂器来估计缺失值
- 采用“代价复杂度剪枝”策略来剪枝
- 对类别不平衡的样本进行处理
- 对于连续值进行处理

缺失值的处理：

替代划分的总的原则就是使其分叉的效果与最佳分叉属性相似，即分叉的误差最小。

$$GINI*(D) = \sum_{i=1}^{k} p_k * (1 - p_k) = 1 - \sum_{i=1}^{k} p_k^2$$

首先，如果某个存在缺失值的特征恰好是当前的分裂增益最大的特征，那么我们需要遍历剩余的特征，剩余的特征中如果有也存在缺失值的特征，那么这些特征忽略，仅仅在完全没有缺失值的特征上进行选择，我们选择其中能够与最佳增益的缺失特征分裂之后增益最接近的特征进行分裂。

如果我们事先设置了一定的标准仅仅选择仅仅选择差异性在一定范围内的特征作为代理特征进行分裂而导致了没有特征和最佳缺失特征的差异性满足要求，或者所有特征都存在缺失值的情况下，缺失样本默认进入个数最大的叶子节点。

显然这种缺失值的处理方式的计算量是非常大的，我们需要遍历其它的特征来进行代理特征选择，这个在数据量很大的情况下开销太大，而带来的性能提升确很有限，所以后来就不怎么用这种处理方式，xgb和lgb中就是直接将缺失值划分到增益大的节点里，这样在处理上要快速的多，而且在gbm的框架下一点点的误差其实影响不大。

连续值的处理：

对于CART分类树连续值的处理问题，其思想和C4.5是相同的，都是将连续的特征离散化。唯一的区别在于在选择划分点时的度量方式不同，C4.5使用的是信息增益，则CART分类树使用的是基尼系数。

类别不平衡的处理：

CART 的一大优势在于： 无论训练数据集有多失衡，它都可以将其子冻消除不需要建模人员采取其他操作。

CART 使用了一种先验机制，其作用相当于对类别进行加权。这种先验机制嵌入于 CART 算法判断分裂优劣的运算里，在 CART 默认的分类模式中，总是要计算每个节点关于根节点的类别频率的比值，这就相当于对数据自动重加权，对类别进行均衡。

对于一个二分类问题，节点 node 被分成类别 1 当且仅当：

比如二分类，根节点属于 1 类和 0 类的分别有 20 和 80 个。 在子节点上有 30 个样本，其中属于 1 类和 0 类的分别是 10 和 20 个。 如果 10/20>20/80，该节点就属于 1 类。

通过这种计算方式就无需管理数据真实的类别分布。 假设有 K 个目标类别，就可以确保根节点中每个类别的概率都是 1/K。 这种默认的模式被称为“先验相等”。

先验设置和加权不同之处在于先验不影响每个节点中的各类别样本的数量或者份额。 先验影响的是每个节点的类别赋值和树生长过程中分裂的选择。

五、研究方向

1．项目介绍（跨语言术语对齐）

关于XGBoost、GBDT、Lightgbm的17个问题：

https://cloud.tencent.com/developer/article/1524927

<https://cloud.tencent.com/developer/article/1524927>