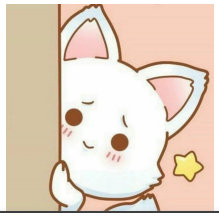
搜索中的 Query 理解及应用

夕小瑶的卖萌屋 5月18日



一只小狐狸带你解锁炼丹术&NLP秘籍

文章作者：Joelchen 腾讯 研究员
编辑整理：Hoh
内容来源：腾讯技术工程出品
链接：https://zhuanlan.zhihu.com/p/112719984

前言

Query 理解（QU，Query Understanding），简单来说就是从词法、句法、语义三个层面对 query 进行结构化解析。这里 query 从广义上来说涉及的任务比较多，最常见的就是我们在搜索系统中输入的查询词，也可以是 FAQ 问答或阅读理解中的问句，又或者可以是人机对话中用户的聊天输入。本文主要介绍在搜索中的 query 理解，会相对系统性地介绍 query 理解中各个重要模块以及它们之间如何 work 起来共同为搜索召回及排序模块服务，同时简单总结个人目前了解到业界在各个模块中的一些实现方法。

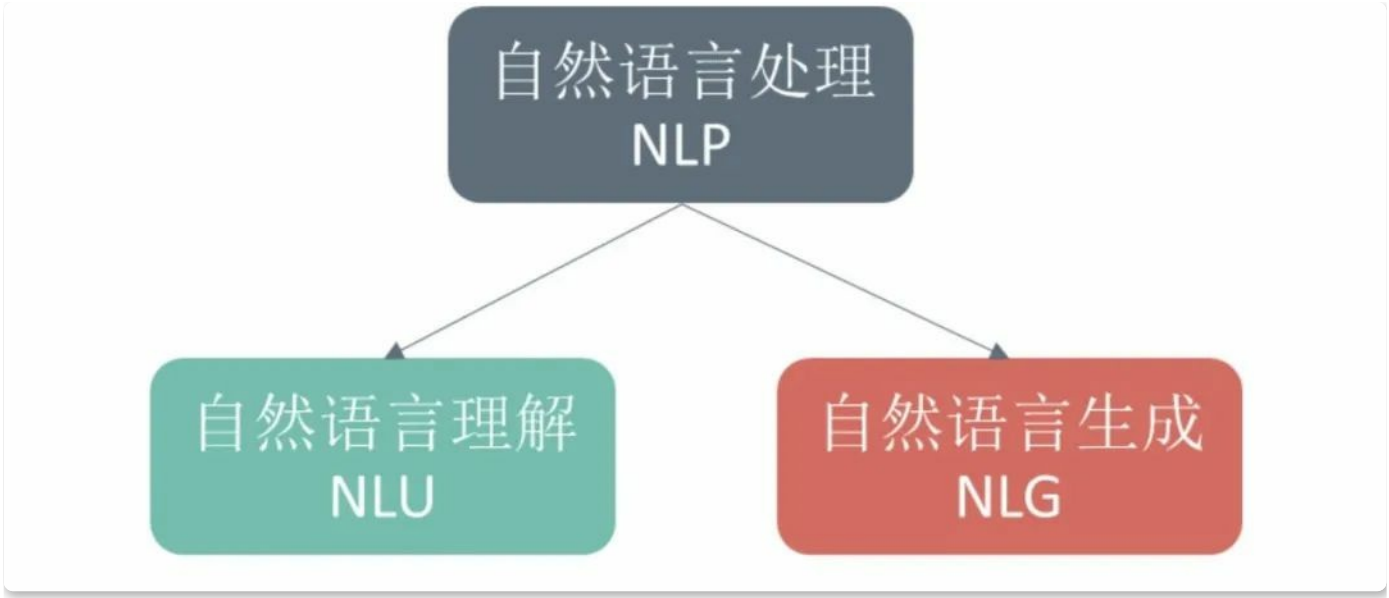
相关概念

1. NLP

自然语言处理（NLP，Natural Language Processing）是集语言学、统计学、计算机科学，人工智能等学科于一体的交叉领域，目标是让计算机能在处理理解人类自然语言的基础上进一步执行结构化输出或语言生成等其他任务，其涉及的基础技术主要有：词法分析、句法分析、语义分析、语用分析、生成模型等。诸如语音识别、机器翻译、QA 问答、对话机器人、阅读理解、文本分类聚类等任务都属于 NLP 的范畴。

这些任务从变换方向上来看，主要可以分为自然语言理解（NLU，Natural Language Understanding）和自然语言生成（NLG，Natural Language Generation）两个方面，其中 NLU 是指对自然语言进行理解并输出结构化语义信息，而 NLG 则是多模态内容（图像、语音、视频、结构/半结构/非结构化文本）之间的相互生成转换。

一些任务同时涵盖 NLU 和 NLG，比如对话机器人任务需要在理解用户的对话内容（NLU 范畴）基础上进行对话内容生成（NLG 范畴），同时为进行多轮对话理解及与用户交互提示这些还需要有对话管理模块（DM，Dialogue Management）等进行协调作出对话控制。本文要介绍的搜索 query 理解大部分模块属于 NLU 范畴，而像 query 改写模块等也会涉及到一些 NLG 方法。



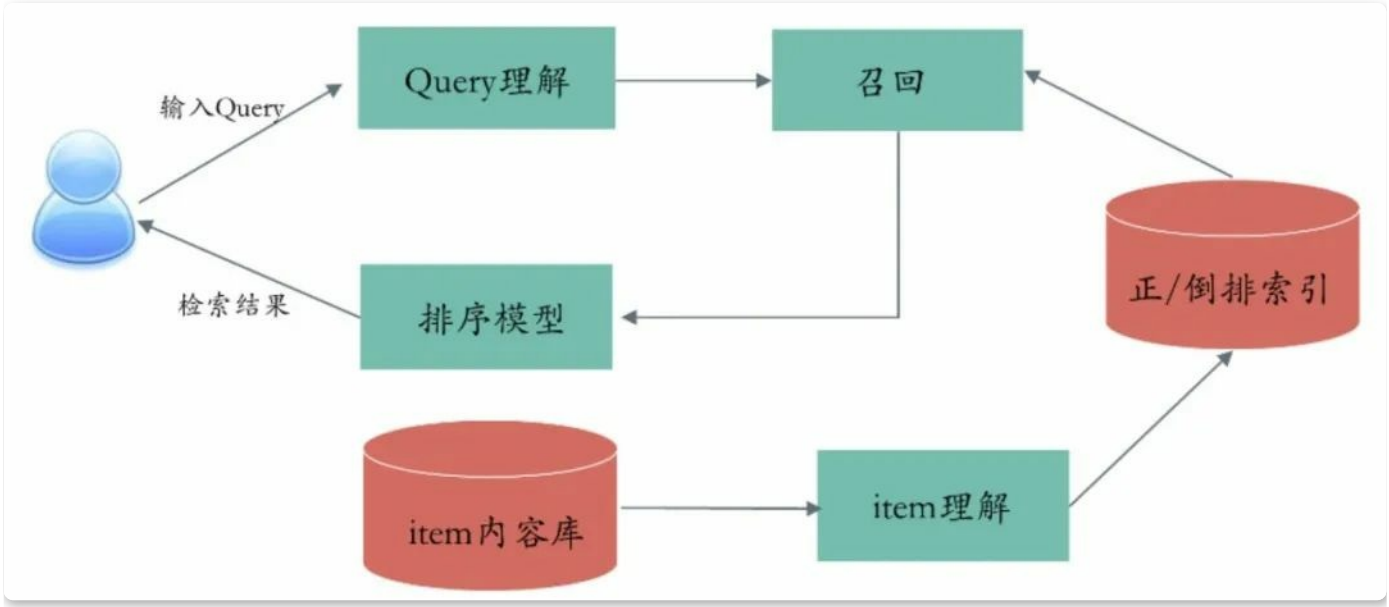
2. 搜索引擎

在介绍搜索 query 理解之前，先简单介绍下搜索引擎相关知识。除了传统的通用搜索 Google、Baidu、Bing 这些，大部分互联网垂直产品如电商、音乐、应用市场、短视频等也都需要搜索功能来满足用户的需求查询，相较于推荐系统的被动式需求满足，用户在使用搜索时会通过组织 query 来主动表达诉求，为此用户的搜索意图相对比较明确。但即便意图相对明确，要做好搜索引擎也是很有挑战的，需要考虑的点及涉及的技术无论在深度还是广度上都有一定难度。

一个基本的搜索系统大体可以分为离线挖掘和在线检索两部分，其中包含的重要模块主要有：Item 内容理解、Query 理解、检索召回、排序模块等。整个检索系统的目标可以抽象为给定 query，检索出最能满足用户需求的 item，也即检索出概率：P(Ii|Q) 最大的 item Ii。根据贝叶斯公式展开：

$$\text{argmax}P\left(I_i|Q\right) = argmax\frac{P(Q|I_i)P(I_i)}{P(Q)}argmxP(Q|I_i)P(I_i)$$

这部分概率对应到基于 query 理解和 item 理解的结果上进行 query 和 item 间相关性计算，同时涉及到点击调权等排序模块。



① 离线挖掘

在离线侧，我们需要做一些基础的离线挖掘工作，包括 item 内容的获取、清洗解析、item 内容理解（语义 tag、权威度计算、时间因子、质量度等）、用户画像构建、query 离线策略挖掘、以及从搜索推荐日志中挖掘 item 之间的语义关联关系、构建排序模型样本及特征工程等。进行 item 内容理解之后，对相应的结构化内容执行建库操作，分别构建正排和倒排索引库。其中，正排索引简单理解起来就是根据 itemid 能找到 item 的各个基本属性及 term 相关（term 及其在 item 中出现的频次、位置等信息）的详细结构化数据。

相反地，倒排索引就是能根据分词 term 来找到包含该 term 的 item 列表及 term 在对应 item 中词频、位置等信息。通常会对某个 item 的 title、keyword、anchor、content 等不同属性域分别构建倒排索引，同时一般会根据 item 资源的权威度、质量度等纵向构建分级索引库，首先从高质量库中进行检索优先保证优质资源能被检索出来，如果高质量检索结果不够再从低质量库中进行检索。为了兼顾索引更新时效性和检索效率，一般会对索引库进行横向分布式部署，且索引库的构建一般分为全量构建和增量更新。常见的能用于快速构建索引的工具框架有 Lucene 全文检索引擎以及基于 Lucene 的 Solr、ES（Elastic Search）等。

除了基本的文本匹配召回，还需要通过构建 query 意图 tag 召回或进行语义匹配召回等多路召回来提升搜索语义相关性以及保证召回的多样性。

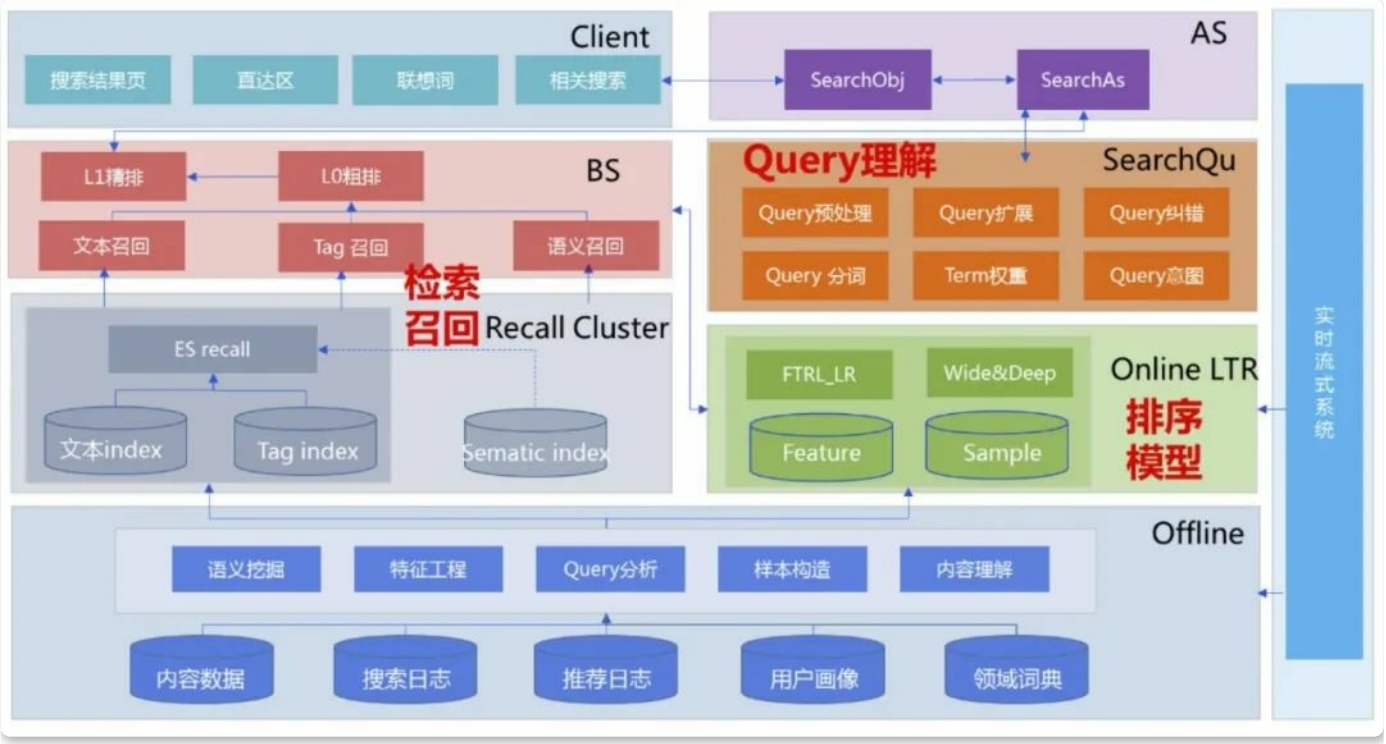
② 在线检索

线上执行检索时大体可以分为基础检索（BS）和高级检索（AS）两个过程，其中 BS 更注重 term 级别的文本相关性匹配及粗排，AS 则更注重从整体上把控语义相关性及进行精排等处理。以下面这个框图为例，介绍下一个相对简单的在线检索过程。对于从 client 发起的线上搜索请求，会由接入层传入 SearchObj（主要负责一些业务逻辑的处理），再由 SearchObj 传给 SearchAS（负责协调调用 qu、召回和排序等模块），SearchAS 首先会去请求 SearchQU 服务（负责搜索 query 理解）获取对 query 理解后的结构化数据，然后将这些结构化数据传给基础召回模块 BS，BS 根据切词粒度由粗到细对底层索引库进行一次或多次检索，执行多个索引队列的求交求并拉链等操作返回结果。

同时 BS 还需要对文本、意图 tag、语义召回等不同路召回队列根据各路召回特点采用多个相关性度量（如：BM25 文本相似度、tag 相似度、语义相关度、pagerank 权威度、点击调权等）进行 L0 粗排截断以保证相关性，然后再将截断后的多路召回进行更精细的 L1 相关性融合排序，更复杂一些的搜索可能会有 L0 到 LN 多层排序，每层排序的侧重点有所不同，越高层次 item 数变得越少，相应的排序方法也会更复杂。

BS 将经过相关性排序的结果返回给 SearchAS，SearchAS 接着调用 SearchRank 服务进行 ctr/cvr 预估以对 BS 返回的结果列表进行 L2 重排序，并从正排索引及摘要库等获取相应 item 详情信息进一步返回给 SearchObj 服务，与此同时 SearchObj 服务可以异步去请求广告服务拉取这个 query 对应的广告召回队列及竞价相关信息，然后就可以对广告或非广告 item 内容进行以效果（pctr、pcvr、pcpm）为导向的排序从而确定各个 item 内容的最终展示位置。

最后在业务层一般还会支持干预逻辑以及一些规则策略来实现各种业务需求。在实际设计搜索实验系统时一般会对这些负责不同功能的模块进行分层管理，以上面介绍的简单检索流程为例主要包括索引召回层、L0 排序截断层、L1 相关性融合排序层、L2 效果排序层、SearchAS 层、业务层等，各层流量相互正交互不影响，可以方便在不同层进行独立的 ABtest 实验。



当然，具体实现搜索系统远比上面介绍的流程更为复杂，涉及的模块及模块间的交互会更多，比如当用户搜索 query 没有符合需求的结果时需要做相关搜索 query 推荐以进行用户引导、检索无结果时可以根据用户的画像或搜索历史做无结果个性化推荐等，同时和推荐系统一样，搜索周边涉及的系统服务也比较庞杂，如涉及日志上报回流、实时计算能力、离/在线存储系统、ABtest 实验平台、报表分析平台、任务调度平台、机器学习平台、模型管理平台、业务管理后台等，只有这些平台能 work 起来，整个搜索系统才能正常运转起来。

③ 多模语义搜索

随着资源越来越丰富，如何让用户从海量的信息及服务资源中能既准确又快速地找到诉求变得越来越重要，如2009年百度提出的框计算概念目标就是为了能更快更准地满足用户需求。同时，随着移动互联网的到来，用户的输入方式越来越多样，除了基本的文本输入检索，实现通过语音、图片、视频等内容载体进行检索的多模态搜索也变得越来越有必要。由于用户需求的多样或知识背景不同导致需求表达的千差万别，要做到极致的搜索体验需要漫长的优化过程，毕竟目前真正的语义搜索还远未到来。

Query 理解

从前面的介绍可以知道，搜索三大模块的大致调用顺序是从 query 理解到检索召回再到排序，作为搜索系统的第一道环节，query 理解的结果除了可以用于召回也可以给排序模块提供必要的基础特征，为此 QU 很大程度影响召回和排序的质量，对 query 是进行简单字面上的理解还是可以理解其潜在的真实需求很大程度决定着一个搜索系统的智能程度。目前业界的搜索 QU 处理流程还是以 pipeline 的方式为主，也即拆分成多个模块分别负责相应的功能实现，pipeline 的处理流程可控性比较强，但存在缺点就是其中一个模块不 work 或准确率不够会对全局理解有较大影响。为此，直接进行 query-item 端到端地理解如深度语义匹配模型等也是一个值得尝试的方向。

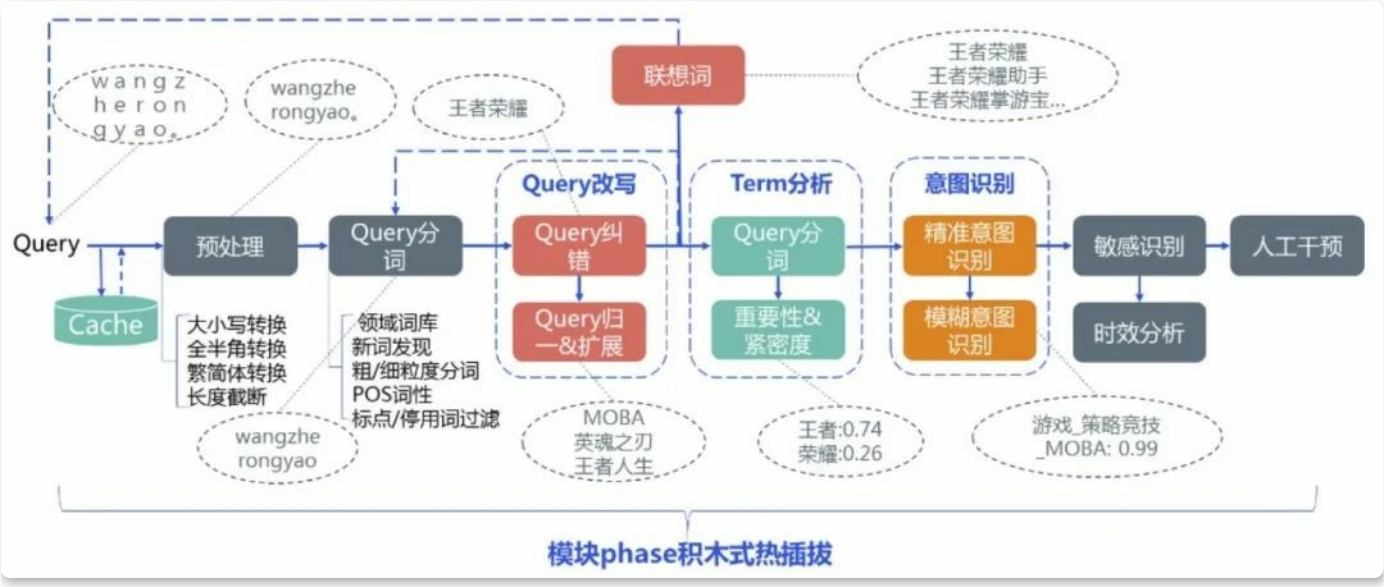
1. Pipeline 流程

搜索 Query 理解包含的模块主要有：query 预处理、query 纠错、query 扩展、query 归一、联想词、query 分词、意图识别、term 重要性分析、敏感 query 识别、时效性识别等。以下图为例，这里简单介绍下 query 理解的 pipeline 处理流程：线上来一个请求

query，为缓解后端压力首先会判断该 query 是否命中 cache，若命中则直接返回对该 query 对应的结构化数据。若不命中 cache，则首先会对 query 进行一些简单的预处理，接着由于 query 纠错可能会用到分词 term 信息进行错误检测等先进行 query 分词并移除一些噪音符号，然后进行 query 纠错处理，一些情况下局部纠错可能会影响到上下文搭配的全局合理性，为此还可能会需要进行二次纠错处理。

对 query 纠错完后可以做 query 归一、扩展及联想词用于进行扩召回及帮助用户做搜索引导。接着再对 query 做分词及对分词后的 term 做重要性分析及紧密度分析，对无关紧要的词可以做丢词等处理，有了分词 term 及对应的权重、紧密度信息后可以用于进行精准和模糊意图的识别。除了这些基本模块，有些搜索场景还需要有对 query 进行敏感识别及时效性分析等其他处理模块。最后还需要能在 cms 端进行配置的人工干预模块，对前面各个模块处理的结果能进行干预以快速响应 badcase 处理。

当然，这个 pipeline 不是通用的，不同的搜索业务可能需要结合自身情况做 pipeline 的调整定制，像百度这些搜索引擎会有更为复杂的 query 理解 pipeline，各个模块间的依赖交互也更为复杂，所以整个 QU 服务最好能灵活支持各个模块的动态热插拔，像组装乐高积木一样进行所需模块的灵活配置，下面对各个模块做进一步详细的介绍。



2. Query 预处理

Query 预处理这个模块相对来说比较简单，主要对 query 进行以下预处理从而方便其他模块进行分析：

- 全半角转换：将在输入法全角模式下输入的 query 转换为半角模式的，主要对英文、数字、标点符号有影响，如将 "ｗｅｃｈａｔ１２３" 全角输入转成半角模式下的 "wechat 123"。
- 大小写转换：统一将大写形式的字母转成小写形式的。
- 繁简体转换：将繁体输入转成简体的形式，当然考虑到用户群体的差异以及可能存在繁体形式的资源，有些情况还需要保留转换前的繁体输入用于召回。
- 无意义符号移除：移除诸如火星文符号、emoji 表情符号等特殊符号内容。
- Query 截断：对于超过一定长度的 query 进行截断处理。

3. Query 分词

① 分词技术

Query 分词就是将 query 切分成多个 term，如："手机淘宝"切分成"手机 淘宝"两个 term。分词作为最基础的词法分析组件，其准确性很大程度影响 QU 后面的各个处理，如分词及对应词性信息可以用于后续的 term 重要性分析和意图识别等多个模块。同时，QU 的分词及其粒度需要与 item 侧索引构建的分词及粒度保持一致，从而才能进行有效地召回。目前分词技术相对来说已经比较成熟，主要做法有基于词典进行前后向最大匹配、对所有成词情况构造 DAG、hmm/crf 序列标注模型以及深度学习模型+序列标注等。

目前无论学术界还是工业界开放的分词工具或服务还是比较多的，如主要有腾讯内部的 QQSeg、百度 LAC、Jieba 分词、清华 THULAC、北大 pkuseg、中科院 ICTCLAS、哈工大 PyLTP、复旦 FNLP、Ansj、SnowNLP、FoolNLTK、HanLP、斯坦福 CoreNLP、Jiagu、IKAnalyzer 等。这些分词工具在功能和性能上会存在一定的差异，具体使用时要结合业务需求定制选择，需要考虑的点主要包括：切词准确性、粒度控制、切词速度、是否带有 NER、NER 识别速度、是否支持自定义词典等，由于没对所有这些分词工具做各项性能指标的具体评测，这里暂时不做过多的比较。

在搜索中的 query 切词一般会做粒度控制，分为细粒度和 phrase 粗粒度两个级别，比如 query "下载深海大作战"按 phrase 粗粒度切分可以为"下载 深海大作战"，按细粒度切分为"下载 深海 大 作战"。在进行召回时可以优先用 phrase 粗粒度的切词结果进行召回能得到更精准相关的结果同时减少多个 term 拉链合并的计算量。当 phrase 粗粒度分词进行召回结果不够时，可以采用拆分后的细粒度分词进行二次重查扩召回。

② 新词发现

一般来说，使用已有的开源切词工具已经有比较好的切分精度了，但是对于一些新出现的网络词汇可能不能及时识别覆盖，尤其是对于一些垂直搜索有比较多业务专名词的情况，这时候需要对这些未登录词做新词发现。一些切词工具自带有新词发现功能，比如 Jieba 采用 HMM 进行新词发现。此外简单地，可以采用基于统计的方法来进行新词发现，通过统计语料中的词语 tfidf 词频、凝聚度和自由度等指标来进行无监督新词挖掘，当词语的凝聚度和自由度达到一定阈值且已有分词不能切分到一起时可以人工评估后加进词库。其中凝聚度即点互信息：

$$Solid = log\frac{P(w_i w_j)}{P(w_i)P(w_j)}$$

还有的做法是对 query 进行切词后构建词之间的关系矩阵，然后进行矩阵分解，得到各个词在主特征空间的投影向量，通过投影向量计算词之间的相似度并设定相应阈值构造0-1切分矩阵，基于该矩阵进行是否成词的判断。再有就是可以将新词发现转化为序列标注问题，训练 BiLSTM-CRF、BERT-CRF 等模型预测成词的起始、中间及结束位置，或者转化为 ngram 词在给定句子语境中是否成词或不成词二分类问题。

4. 紧密度分析

Term 紧密度，主要用于衡量 query 中任意两个 term 之间的紧密程度，如果两个 term 间紧密度比较高，则这两个 term 在召回 item 中出现的距离越近相对来说越相关。以相邻的两个 term 为例，由于分词工具本身存在准确率问题，某两个 term 可能不会被分词工具切分到一起，但它们之间的关系比较紧密，也即紧密度比较高，如果在召回时能将在 item 中同时出现有这两个 term 且出现位置挨在一起或比较靠近的 item 进行召回，相关性会更好。

以前面的搜索 query "下载深海大作战"为例，经分词工具可能切分成"下载 深海 大 作战"，但其实"大"和"作战"的紧密度很高，从文本相关性角度来看，召回"喵星大作战" app 要一定程度比"大人物作战"会更相关。当然，在 query 中并不是两个 term 的距离越近紧密度越高，可以通过统计搜索 log 里 term 之间的共现概率来衡量他们的紧密程度。

在进行召回时，传统的相关性打分公式如 OkaTP、BM25TP、newTP、BM25TOP 等在 BM25 基础上进一步考虑了 proximity 计算，但主要采用两个 term 在 item 中的距离度量，如：

$$\operatorname{tpi}(t_i, t_j) = \frac{1.0}{\operatorname{dist}[t_i, t_j]^2}$$

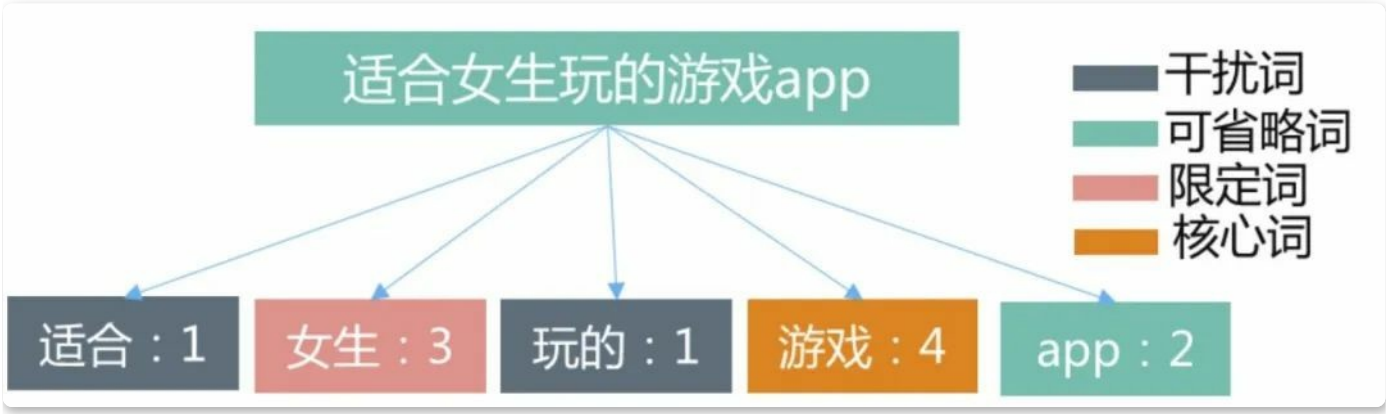
有了 query 中的 term 紧密度，在召回构造查询索引的逻辑表达式中可以要求紧密度高的两个 term 需共同出现以及在 proximity 计算公式中融合考虑进去，从而保证 query 中紧密度高的两个 term 在 item 中出现距离更近更相关。

5. Term 重要性分析

考虑到不同 term 在同一文本中会有不一样的重要性，在做 query 理解及 item 内容理解时均需要挖掘切词后各个 term 的重要性，在进行召回计算相关性时需要同时考虑 query 及 item 侧的 term 重要性。如对于 query "手机淘宝"，很明显 term "淘宝"要比"手机"更重要，为此赋予"淘宝"的权重应该比"手机"高。Term 重要性可以通过分等级或0.0~1.0的量化分值来衡量，如下图的 case 所示我们可以将 term 重要性分为4个级别，重要性由高到低分别是：核心词、限定词、可省略词、干扰词。对于重要级别最低的 term 可以考虑直接丢词，或者在索引库进行召回构造查询逻辑表达式时将对应的 term 用 "or" 逻辑放宽出现限制，至于计算出的在 query 和 item 内容中的 term 重要性分值则可以用于召回时计算基础相关性，如简单地将 BM25 公式：

$$\sum_{t \in Q} \log \frac{N - n(t) + 0.5}{n(t) + 0.5} \times \frac{tf(t)(k_1 + 1)}{\operatorname{tf}(t) + k_1\left(1 - b + b\frac{\mathrm{dl}}{\mathrm{avgdl}}\right)} \frac{qf(t)(k_2 + 1)}{\operatorname{qf}(t) + k_2}$$

中 term 在 item 及 query 中的词频 tf(t)、qf(t) 乘上各自的 term 权重。



其中 item 内容侧的 term 重要性可以采用 LDA 主题模型、TextRank 等方法来挖掘，至于 query 侧的 term 重要性，比较容易想到的方法就是把它视为分类或回归问题来考虑，通过训练 svm、gbdt 等传统机器学习模型即可进行预测。模型样本的构造除了进行人工标注还可以通过用户的搜索点击日志来自动构造。大概做法是将某个 query 对应搜索结果的用户点击频次作为同时出现在 query 及搜索结果 title 中相应 term 的重要性体现，首先通过共同点击信息或二部图方法对 query 进行聚类，再将一个 query 对应有不同点击 title 以及同一 term 在簇内不同 query 中的点击 title 频次结果加权考虑起来，同时排除掉一些搜索 qv 比较不置信的 query 及 title 对应的结果，最后将累计频次进行归一化及分档得到样本对应 label。

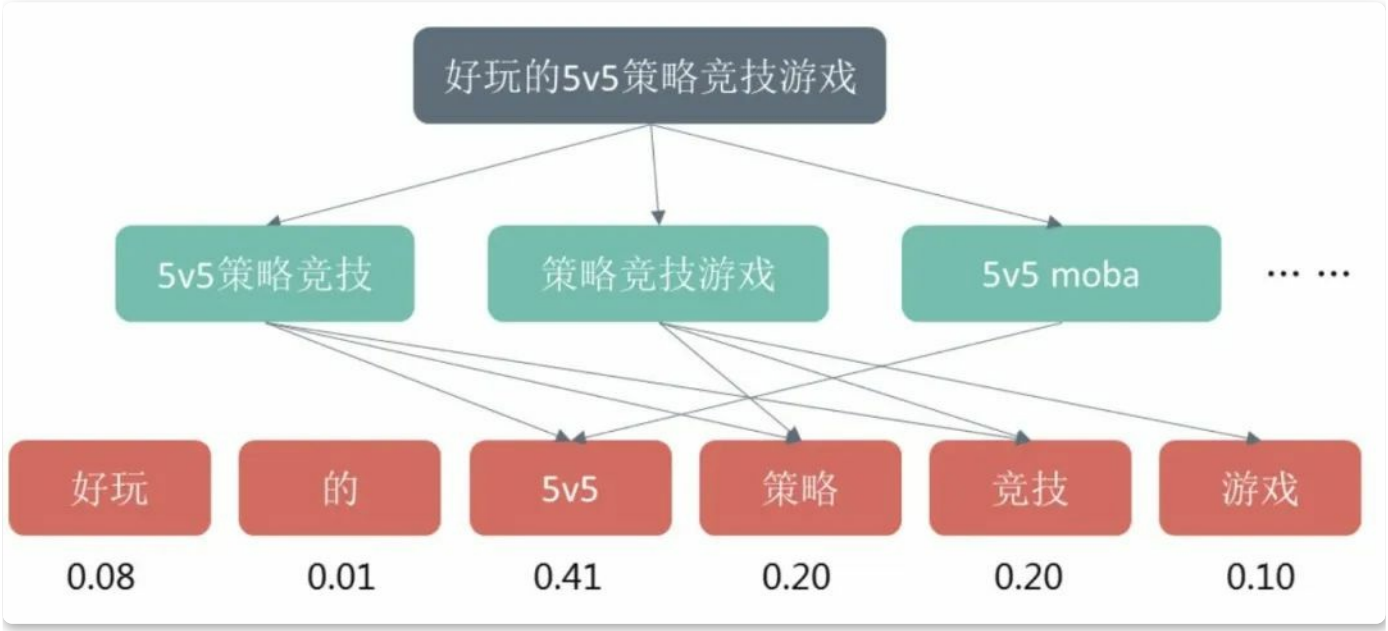
至于特征方面，则可以从词法、句法、语义、统计信息等多个方向进行构造，比如：term 词性、长度信息、term 数目、位置信息、句法依存 tag、是否数字、是否英文、是否停用词、是否专名实体、是否重要行业词、embedding 模长、删词差异度、前后词互信息、左右邻熵、独立检索占比 (term 单独作为 query 的 qv / 所有包含 term 的 query 的 qv 和）、iqf、文档 idf、统计概率：

$$P(t|d) = \frac{P(d|t) * P(t)}{P(d)}$$

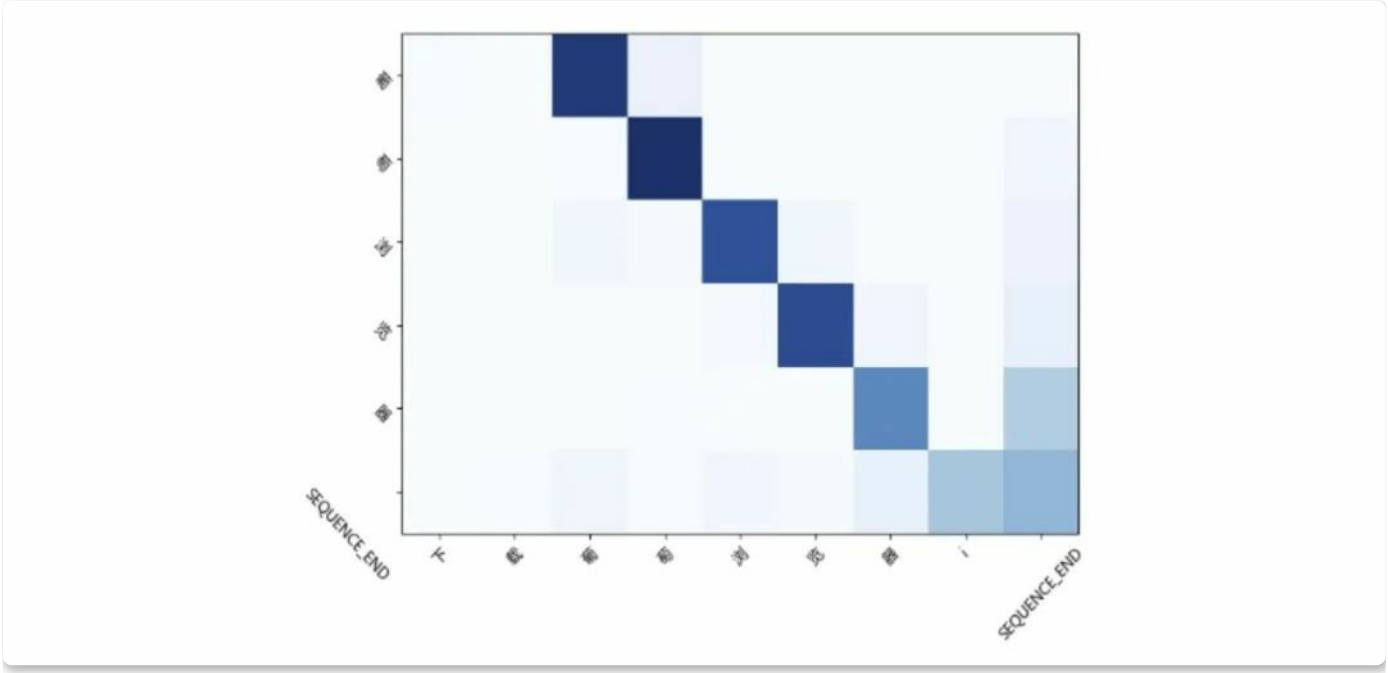
以及短语生成树得到 term 权重等。

其中删词差异度的做法是先训练得到 query 的 embedding 表示，然后计算移除各个 term 之后的 query 与原 query 的 embedding 相似度差值用于衡量 term 的重要性，如果移除某个 term 之后相似度差异很大，代表这个 term 比较重要。而短语生成树的做法是通过从搜索 session 序列及搜索点击行为中挖掘出 query 之间的相关关系按 query 长度降序自顶向下进行构造得到，其中树的边和结点均有一定的权重。

这里假设在一定共现关系情况下越短的 query 越是整体意图的核心表达，所以和越下层结点连接越多的 term 重要性越大，仅和较上层或根结点有连接的 term 重要性相对较小，通过用 iqf 等初始化叶子结点，然后自底向上进行结点分值计算并进行多轮迭代使得 term 权重相对稳定。如下图所示 query "好玩的5v5策略竞技游戏"构造的短语生成树示例。



分类回归的方法很好地利用了用户的点击行为反馈，通过精细的特征工程往往能得到比较好的结果。还有的方法就是利用深度学习模型来学习 term 重要性，比如通过训练基于 BiLSTM+Attention 的 query 意图分类模型或基于 eq2Seq/Transformer 训练的 query 翻译改写模型得到的 attention 权重副产物再结合其他策略或作为上述分类回归模型的特征也可以用于衡量 term 的重要性。



6. 搜索引导

受限于用户的先验知识的参差不齐或输入设备引入的噪音，用户输入的 query 可能不足以表达用户真正的需求进而影响用户搜到想要的结果。为此，除了保证搜索结果的相关性，一个完善的搜索引擎还需要给用户提供一系列搜索引导功能，以减少用户的搜索输入成本，缩短用户找到诉求的路径。搜索引导按功能的不同主要可以分为：搜索热词、搜索历史、query 改写、搜索联想词，一些电商等垂搜可能还带有类目属性等搜索导航功能。由于搜索热词及搜索历史功能相对比较好理解，这里不做过多阐述。

① Query 改写

Query 改写这个概念比较泛，简单理解就是将源 query 改写变换到另一个 query。按照改写功能的不同，query 改写可以分为 query 纠错、query 归一、query 扩展三个方向。其中 query 纠错负责对存在错误的 query 进行识别纠错，query 归一负责将偏门的 query 归一变换到更标准且同义的 query 表达，而 query 扩展则负责扩展出和源 query 内容或行为语义相关的 query 列表推荐给用户进行潜在需求挖掘发现。

Query 纠错：

Query 纠错，顾名思义，也即对用户输入 query 出现的错误进行检测和纠正的过程。用户在使用搜索过程中，可能由于先验知识掌握不够或输入过程引入噪音（ 如：语音识别有误、快速输入手误等 ）输入的搜索 query 会存在一定的错误。如果不对带有错误的 query 进行纠错，除了会影响 QU 其他模块的准确率，还会影响召回的相关性及排序的合理性，最终影响到用户的搜索体验。

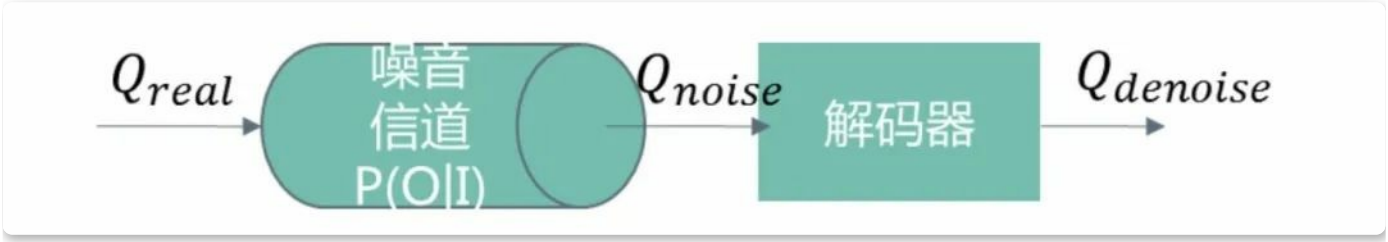
除了搜索场景，query 纠错还可以应用于输入法、人机对话、语音识别、内容审核等应用场景。不同的业务场景需要解决的错误类型会不太一样，比如 ASR 语音识别主要解决同谐音、模糊音等错误，而输入法及搜索等场景则需要面临解决几乎所有错误类型，如同谐音、模糊音（ 平舌翘舌、前后鼻音等 ）、混淆音、形近字（ 主要针对五笔和笔画手写输入法 ）、多漏字等错误。根据 query 中是否有不在词典中本身就有错误的词语（ Non-word ），可以将 query 错误类型主要分为 Non-word 和 Real-word 两类错误。

其中，Non-word 错误一般出现在带英文单词或数字的 query 中，由于通过输入法进行输入，不会存在错误中文字的情况，所以中文 query 如果以字作为最小语义单元的话一般只会存在 Real-word 错误，而带英文数字的 query 则可能存在两类错误。下图对这两大类的常见错误类型进行归类及给出了相应的例子。

从原理上，Query 纠错可以用噪音信道模型来理解，假设用户本意是搜索 Qreal，但是 query 经过噪音信道后引进了一定的噪音，此时纠错过程相当于构建解码器将带有噪音干扰的 query Qnoise 进行最大去噪还原成 Qdenoise，使得 Qdenoise≈Qreal。对应的公式为：

$$Q_{\text{real}} = \text{argmax}_{Q_{\text{denoise}}} P(Q_{\text{denoise}}|Q_{\text{noise}}) = \text{argmax}_{Q_{\text{denoise}}} \frac{P(Q_{\text{noise}}|Q_{\text{denoise}}) \times P(Q_{\text{denoise}})}{P(Q_{\text{noise}})} \propto \text{argmax}_{Q_{\text{denoise}}} P(Q_{\text{noise}}|Q_{\text{denoise}}) \times P(Q_{\text{denoise}})$$

其中 P(Qdenoise) 表示语言模型概率，P(Qnoise|Qdenoise) 表示写错概率，进行纠错一般都是围绕着求解这两个概率来进行的。



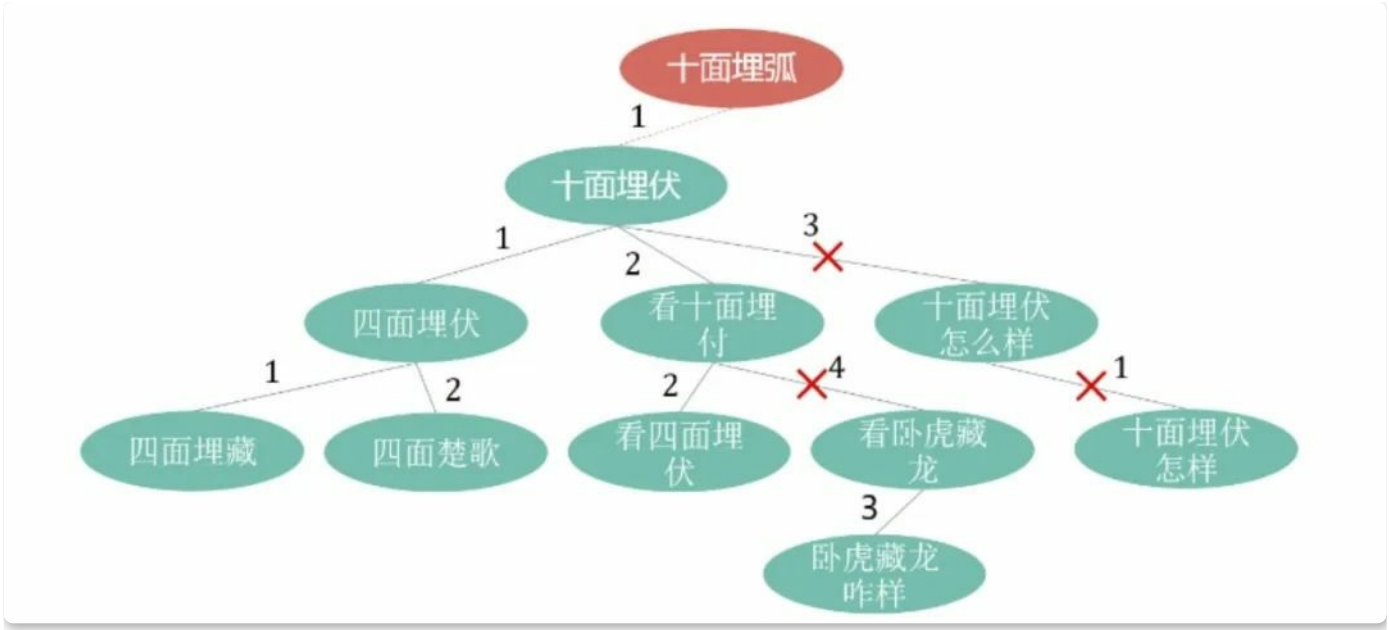
纠错任务主要包含错误检测和错误纠正两个子任务，其中错误检测用于识别错误词语的位置，简单地可以通过对输入 query 进行切分后检查各个词语是否在维护的自定义词表或挖掘积累的常见纠错 pair 中，若不在则根据字型、字音或输入码相近字进行替换构造候选并结合 ngram 语言模型概率来判断其是否存在错误，这个方法未充分考虑到上下文信息，可以适用于常见中文词组搭配、英文单词错误等的检测。进一步的做法是通过训练序列标注模型的方法来识别错误的开始和结束位置。

至于错误纠正，即在检测出 query 存在错误的基础上对错误部分进行纠正的过程，其主要包括纠错候选召回、候选排序选择两个步骤。在进行候选召回时，没有一种策略方法能覆盖所有的错误类型，所以一般通过采用多种策略方法进行多路候选召回，然后在多路召回的基础上通过排序模型来进行最终的候选排序。

对于英文单词错误、多漏字、前后颠倒等错误可以通过编辑距离度量进行召回，编辑距离表示从一个字符串变换到另一个字符串需要进行插入、删除、替换操作的次数，如 "apple" 可能错误拼写成 "appel"，它们的编辑距离是1。由于用户的搜索 query 数一般是千万甚至亿级别的，如果进行两两计算编辑距离的话计算量会非常大，为此需要采用一定的方法减小计算量才行。比较容易想到的做法是采用一些启发式的策略，如要求首字（符）一样情况下将长度小于等于一定值的 query 划分到一个桶内再计算两两 query 间的编辑距离，此时可以利用 MapReduce 进一步加速计算。

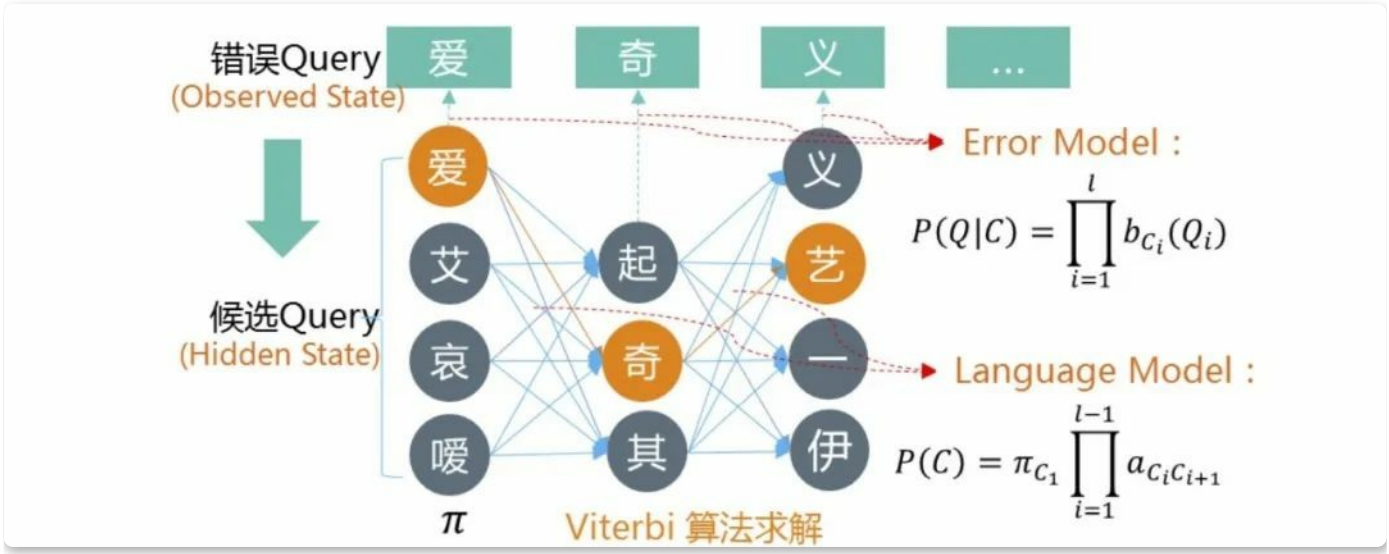
当然这种启发式的策略可能会遗漏掉首字（符）不一样的 case，如在前面两个位置的多漏字、颠倒等错误。还有的办法就是利用空间换时间，如对 query 进行 ngram 等长粒度切分后构建倒排索引，然后进行索引拉链的时候保留相似度 topN 的 query 作为候选。又或者利用编辑距离度量满足三角不等式 d(x,y)+d(y,z)≥d(x,z) 的特性对多叉树进行剪枝来减少计算量。

首先随机选取一个 query 作为根结点，然后自顶向下对所有 query 构建多叉树，树的边为两个结点 query 的编辑距离。给定一个 query，需要找到与其编辑距离小于等于 n 的所有 query，此时自顶向下与相应的结点 query 计算编辑距离 d，接着只需递归考虑边值在 d-n 到 d+n 范围的子树即可。如下图所示需要查找所有与 query "十面埋弧"编辑距离小于等于1的 query，由于"十面埋弧"与"十面埋伏"的编辑距离为1，此时只需考虑边值在1-1到1+1范围的子树，为此"十面埋伏怎么样"作为根结点的子树可以不用继续考虑。



对于等长的拼音字型错误类型还可以用 HMM 模型进行召回，HMM 模型主要由初始状态概率、隐藏状态间转移概率及隐藏状态到可观测状态的发射概率三部分组成。如下图所示，将用户输入的错误 query "爱奇义"视为可观测状态，对应的正确 query "爱奇艺"作为隐藏状态，其中正确 query 字词到错误 query 字词的发射关系可以通过人工梳理的同谐音、形近字混淆词表、通过编辑距离度量召回的相近英文单词以及挖掘好的纠错片段对得到。

至于模型参数，可以将搜索日志中 query 进行采样后作为样本利用 hmmlearn、pomegranate 等工具采用 EM 算法进行无监督训练，也可以简单地对搜索行为进行统计得到，如通过 nltk、srilm、kenlm 等工具统计搜索行为日志中 ngram 语言模型转移概率，以及通过统计搜索点击日志中 query-item 及搜索 session 中 query-query 对齐后的混淆词表中字之间的错误发射概率。训练得到模型参数后，采用维特比算法对隐藏状态序列矩阵进行最大纠错概率求解得到候选纠错序列。



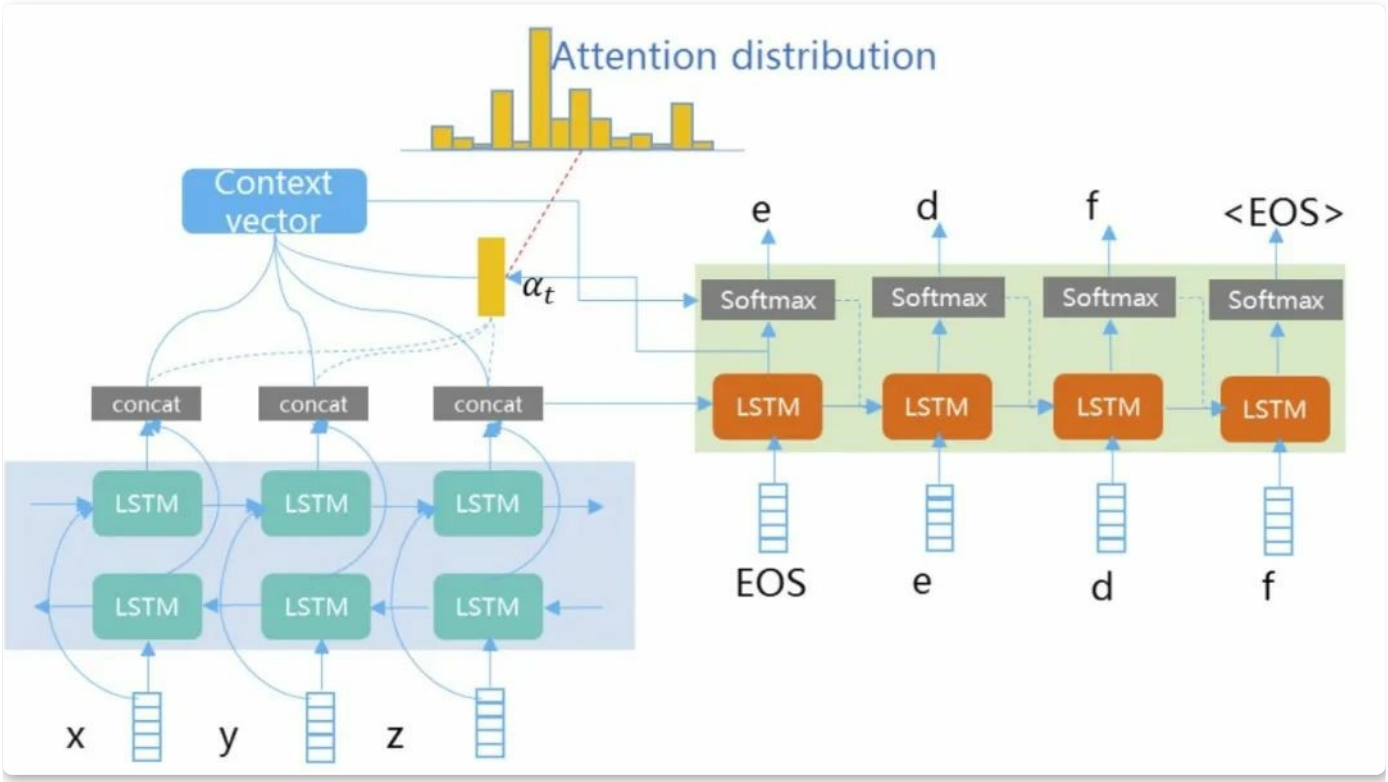
$$C^* = \underset{c}{argmax} P(C|Q) = \underset{c}{argmax} \frac{P(C,Q)}{P(Q)} = \underset{c}{argmax} (\pi_{C_1} \prod_{i=1}^{l-1} a_{C_i C_{i+1}} \prod_{i=1}^{l} b_{C_i}(Q_i))$$

进一步地，我们还可以尝试深度学习模型来充分挖掘搜索点击行为及搜索 session 进行纠错候选召回，如采用 Seq2Seq、Transformer、Pointer-Generator Networks 等模型进行端到端的生成改写，通过引入 attention、copy 等机制以及结合混淆词表进行受限翻译生成等优化，可以比较好地结合上下文进行变长的候选召回。

另外结合 BERT 等预训练语言模型来进行候选召回也是值得尝试的方向，如在 BERT 等预训练模型基础上采用场景相关的无监督语

料继续预训练，然后在错误检测的基础上对错误的字词进行 mask 并预测该位置的正确字词。Google 在2019年提出的 LaserTagger 模型则是另辟蹊径将文本生成建模为序列标注任务，采用 BERT 预训练模型作为 Encoder 基础上预测各个序列位置的增删留标签，同样适用于 query 纠错这种纠错前后大部分文本重合的任务。

另外，爱奇艺在同一年提出的适用于繁简体中文拼写检纠错的 FASPell 模型尝试在利用 BERT 等预训练语言模型生成纠错候选矩阵（DAE）的基础上结合生成候选的置信度及字符相似度对候选进行解码过滤（CSD）出候选纠错目标，在中文纠错任务上也取得了一定进展。



在多路召回的基础上，可以通过从词法、句法、语义及统计特征等多个方面构造特征训练 GBDT、GBRank 等排序模型将预测出最优的纠错候选。由于进行多路纠错候选召回计算量相对比较大且直接进行线上纠错会存在较大的风险，可以在离线挖掘好 query 纠错 pair 后按搜索 qv 优先进行对头中部 query 进行人工审核准入，然后放到线上存储供调用查询。

线上应用时，当 QU 识别到用户输入 query 存在错误并进行相应纠错后，对于不那么置信的结果可以将纠错结果透传给召回侧进行二次检索使用，对于比较置信的纠错结果可以直接展示用纠错后 query 进行召回的结果并给到用户搜索词确认提示（如下图所示），如果纠错 query 不是用户真实意图表达的话，用户可以继续选择用原 query 进行搜索，此时用户的反馈行为也可以用于进一步优化 query 纠错。

Query 扩展：

Query 扩展，即通过挖掘 query 间的语义关系扩展出和原 query 相关的 query 列表。Query 列表的结果可以用于扩召回以及进行

query 推荐帮用户挖掘潜在需求，如下图在百度搜索"自然语言处理"扩展出的相关搜索 query：

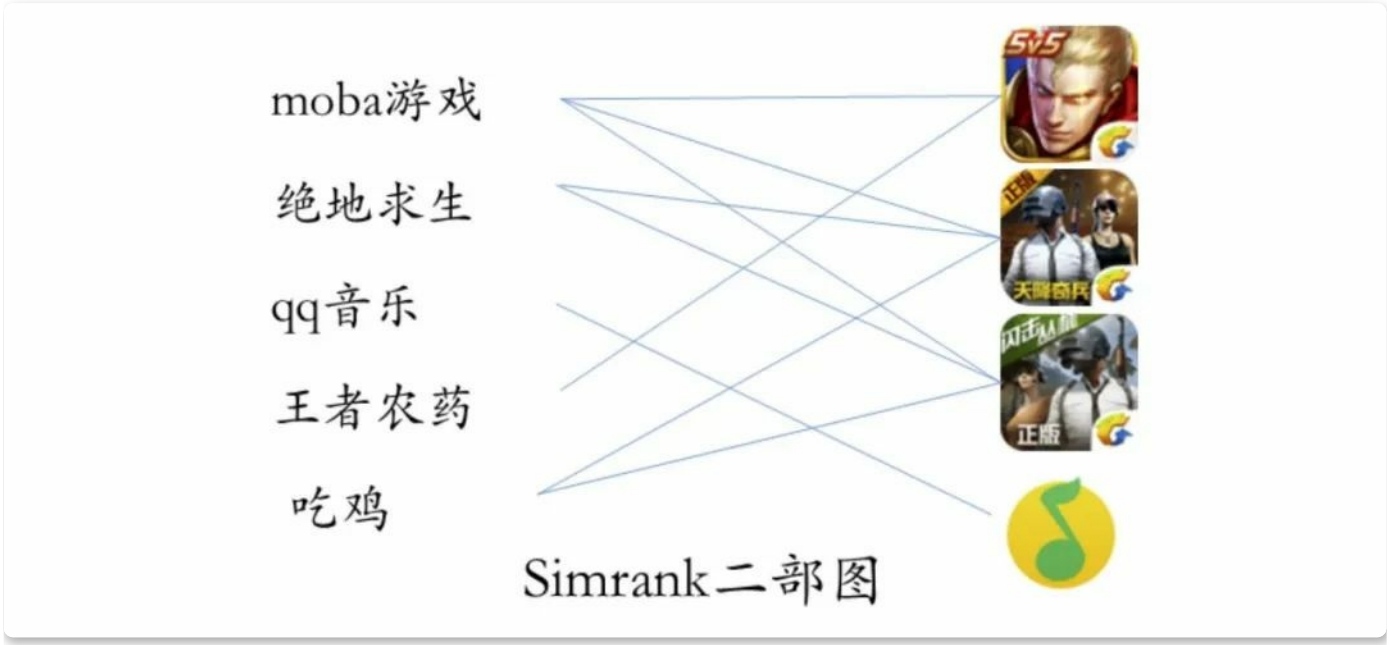
搜索场景有丰富的用户行为数据，我们可以通过挖掘搜索 session 序列和点击下载行为中 query 间的语义关系来做 query 扩展。如用户在进行搜索时，如果对当前搜索结果不满意可能会进行一次或多次 query 变换重新发起搜索，那么同一搜索 session 内变换前后的 query 一般存在一定的相关性，为此可以通过统计互信息、关联规则挖掘等方法来挖掘搜索 session 序列中的频繁共现关系。或者把一个用户搜索 session 序列当成文章，其中的每个 query 作为文章的一个词语，作出假设：如果两个 query 有相同的 session 上下文，则它们是相似的，然后通过训练 word2vec、fasttext 等模型将 query 向量化，进而可以计算得到 query 间的 embedding 相似度。

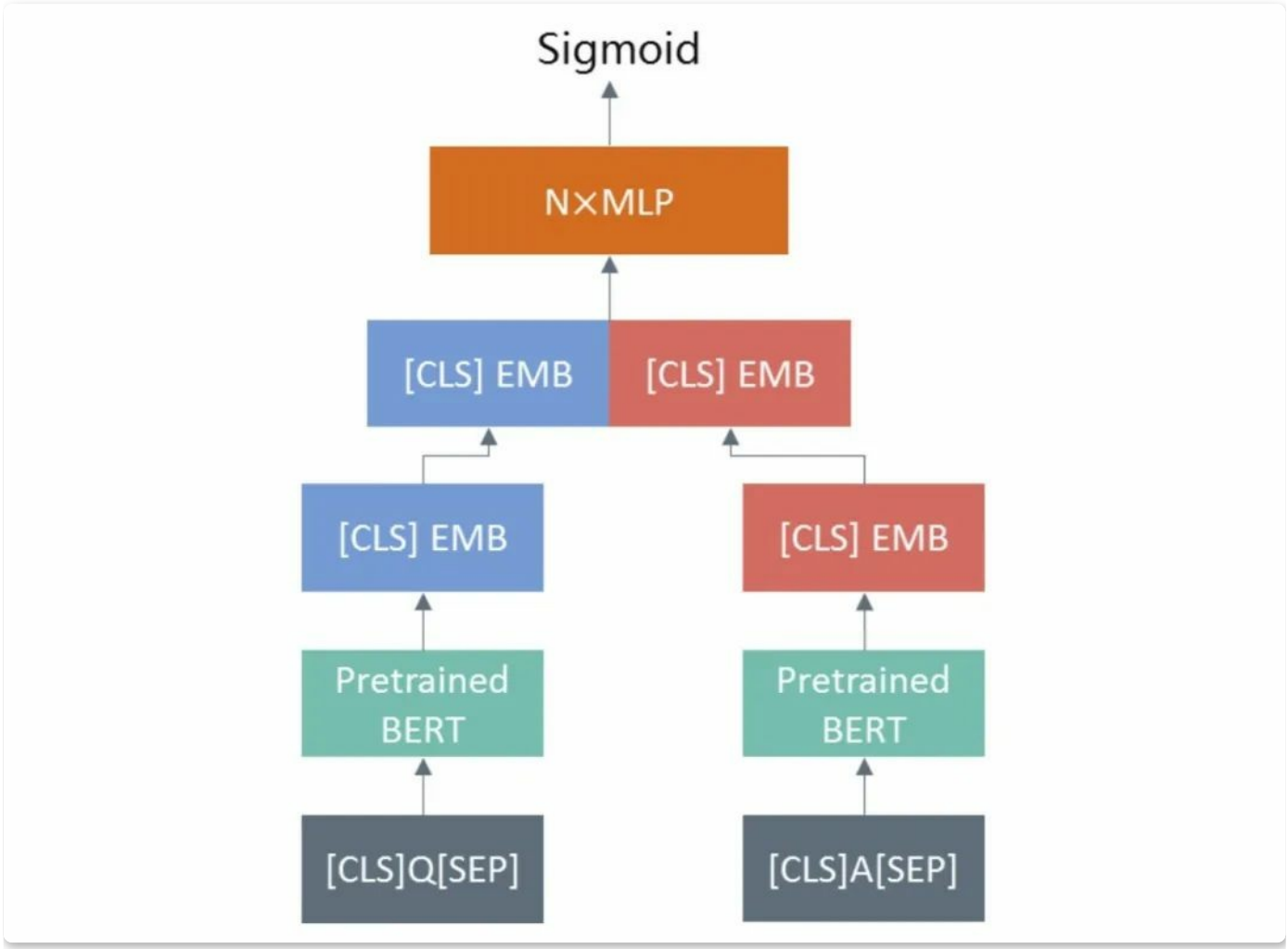
对于长尾复杂 query，通过 word2vec 训练得到的 embedding 可能会存在 oov 的问题，而 fasttext 由于还考虑了字级别的 ngram 特征输入进行训练，所以除了可以得到 query 粒度的 embedding，还可以得到字、词粒度的 embedding，此时通过对未登录 query 切词后的字、词的 embedding 进行简单的求和、求平均也可以得到 query 的 embedding 表示。还可以参考 WR embedding 的做法进一步考虑不同 term 的权重做加权求平均，然后通过减去主成分的映射向量以加大不同 query 间的向量距离，方法简单却比较有效。

至于搜索点击下载行为，可以通过构建 query-item 的点击下载矩阵，然后采用协同过滤或 SVD 矩阵分解等方法计算 query 之间的相似度，又或者构建 query 和 item 之间的二部图（如下图示例），若某个 query 节点和 item 节点之间存在点击或下载行为则用边进行连接，以 ctr、cvr 或归一化的点击下载次数等作为连接边的权重，然后可以训练 swing、simrank/wsimrank++ 等图算法迭代计算 query 间的相似度，也可以采用 Graph Embedding 的方法来训练得到 query 结点间的 embedding 相似度。

更进一步地，我们还可以利用搜索点击下载行为构造弱监督样本训练基于 CNN/LSTM/BERT 等子网络的 query-item 语义匹配模型得到 query 和 item 的 embedding 表示，如此也可以计算 query pair 间的 embedding 相似度。对于将 query 进行 embedding 向量

化的方法，可以先离线计算好已有存量 query 的 embedding 表示，然后用 faiss 等工具构建向量索引，当线上有新的 query 时通过模型 inference 得到对应的 embedding 表示即可进行高效的近邻向量检索以召回语义相似的 query。





在给用户做搜索 query 推荐时，除了上面提到的跟用户当前输入 query 单点相关 query 推荐之外，还可以结合用户历史搜索行为及画像信息等来预测用户当前时刻可能感兴趣的搜索 query，并在搜索起始页等场景进行推荐展示。

此时，可以通过 LSTM 等网络将用户在一段 session 内的搜索行为序列建模为用户 embedding 表示，然后可以通过构建 Encoder-Decoder 模型来生成 query，或采用语义匹配的方法将用户 embedding 及 query embedding 映射到同一向量空间中，然后通过计算

embedding 相似度的方法来召回用户可能感兴趣的 query。

Query 归一：

Query 归一和 query 纠错在概念上容易混淆，相较于 query 纠错是对存在错误的 query 进行纠正，query 归一则主要起到对同近义表达的 query 进行语义归一的作用。一些用户的 query 组织相对来说比较冷门，和 item 侧资源的语义相同但文字表达相差较大，直接用于召回的话相关性可能会打折扣，这时如果能将这些 query 归一到相对热门同义或存在对应资源的 query 会更容易召回相关结果。如将"腾讯台球"归一到"腾讯桌球"，"华仔啥时候出生的?"、"刘德华出生年月"、"刘德华什么是出生的"这些 query 都可以归一到"刘德华出生日期"相对标准的 query。其中涉及到的技术主要有同义词挖掘及语义对齐替换，如"华仔"对应的同义词是"刘德华"，"啥时候出生的"对应的同义词是"出生日期"。

同义词的挖掘是一个积累的过程，最直接的获取方式是利用业界已经有一些比较有名的知识库，如英文版本的 WordNet、中文版本的知网 HowNet、哈工大的同义词词林等，或者可以利用一些开放的中文知识图谱（如：OpenKG、OwnThink 等）或从抓取百度/维基百科站点数据然后提取出其中的别名、简称等结构化信息直接获得，对于百科中无结构化数据可以简单通过一些模板规则（如："XX俗称XX"、"XX又名XX"等）来提取同义词。同时，还可以在知识库中已有同义词种子的基础上通过一些方法进一步扩充同义词，如韩家炜老师团队提出的通过构建分类器来判断实体词是否属于某个同义词簇的方法来进一步扩充同义词集。

除了利用结构化数据或规则模板，还可以在构造平行语料基础上通过语义对齐的方式来挖掘同义词。对于搜索场景来说，可以通过挖掘丰富的行为数据来构造平行语料，如利用搜索 session 行为相关语料训练无监督的 word2vec、wordrank 等词向量模型来衡量词语间的相似度，不过这些模型更多是学习词语间在相同上下文的共现相似，得到的相似度高的词语对不一定是同义词，有可能是同位词、上下位词甚至是反义词，此时需要通过引入监督信号或外部知识来优化词向量，如有方法提出通过构建 multi-task 任务在预测目标词的同时预测目标词在句子中表示的实体类型以加入实体的语义信息来提升词向量之间的语义相似性。

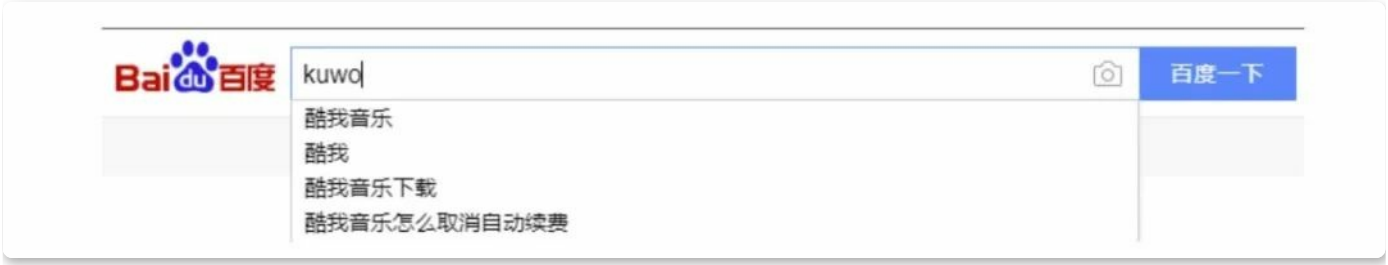
进一步地，还可以利用前面介绍的二部图迭代、深度语义匹配、Seq2Seq 翻译生成等 query 扩展方法从搜索点击弱监督行为中先挖掘出语义表达相近的 query-query、item-item 或 query-item 短语对，然后再将语义相近的 query/item 短语对进行语义对齐，对齐的话可以采用一些规则的方法，也可以采用传统的统计翻译模型如 IBM-M2 进行对齐，语义对齐后从中抽取出处于相同或相近上下文中的两个词语作为同义词对候选，然后结合一些统计特征、词语 embedding 相似度以及人工筛选等方式进行过滤筛选。

考虑到同一个词语在不同的上下文中可能表达不同的语义，同义词语间的关系也是上下文相关的，此时如果通过对齐挖掘粗粒度的同义片段对能进一步消除歧义。线上对 query 进行归一时，则和离线同义词挖掘的过程相反，对 query 进行分词后读取线上存储的同义词表做同义词候选替换，对替换网格进行对齐生成候选 query，最后通过结合语言模型概率及在当前上下文的替换概率或者构造特征训练 GBDT 等模型的方式对候选 query 进行排序得到最终的归一 query。

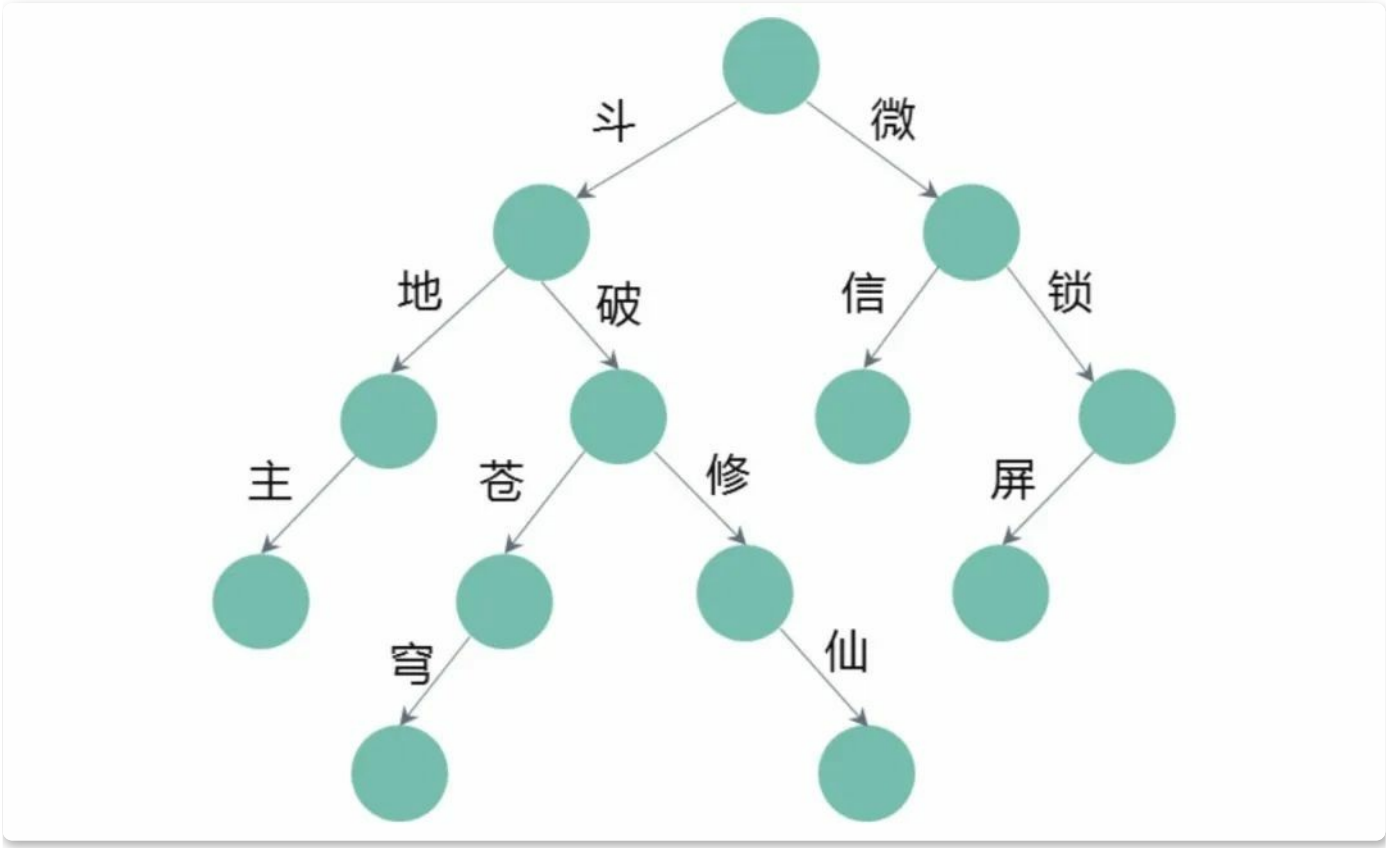
② 搜索联想词

联想词，顾名思义，就是对用户输入 query 进行联想扩展，以减少用户搜索输入成本及输错可能，能比较好地提升用户搜索体验。联想结果主要以文本匹配为主，文本匹配结果不足可以辅以语义召回结果提升充盈率。考虑到用户在输入搜索 query 时意图相对明

确，一般会从左到右进行 query 组织，为此基于这个启发式规则，目前联想词中文本匹配召回又以前缀匹配优先。



虽然联想词涉及的是技术主要是简单的文本匹配，在匹配过程中还需要考虑效率和召回质量，同时中文输入可能会有拼音输入的情况（如上图所示）也需要考虑。由于用户在搜索框中输入每一个字时都会发起一起请求，联想词场景的请求 pv 是非常大的。为加速匹配效率，可以通过对历史搜索 query 按 qv 量这些筛选并预处理后分别构建前后缀 trie 树用于对用户线上输入的 query 进行前缀及中后缀匹配召回，然后对召回的结果进行排序，如果是仅简单按 qv 降序排序，可以在 trie 树结点中存放 qv 信息并用最小堆等结构进行 topK 召回。



当然仅按 qv 排序还不够，比如可能还需要考虑用户输入上文 query 后对推荐的下文 query 的点击转化、下文 query 在结果页的点击转化以及 query 的商业化价值等因素。同时一些短 query 召回的结果会非常多，线上直接进行召回排序性能压力较大，为此可以先通过离线召回并进行粗排筛选，再将召回结果写到一些 kv 数据库如 redis、共享内存等存储供线上直接查询使用。离线召回的话，可以采用 AC 自动机同时进行高效的前中后缀匹配召回。AC 自动机（Aho-Corasic）是基于 trie 数 + KMP 实现的多模匹配算法，可以在目标文本中查找多个模式串的出现次数以及位置。此时，离线召回大致的流程是：

- 从历史搜索 query 中构造前缀 sub-query，如 query "酷我音乐"对应的 sub-query 有中文形式的"酷"、"酷我"、"酷我音"、"酷我音乐"及拼音字符串 "ku"、"kuwo" 等，同时可以加上一些专名实体或行业词，如应用垂搜中的"音乐"、"视频"等功能需求词；
- 利用所有的 sub-query 构建 AC 自动机；
- 利用构建的 AC 自动机对历史搜索 query 及其拼音形式分别进行多模匹配，从中匹配出所有的前中后缀 sub-query，进而得到 <sub-query，query> 召回候选。

- 按照一定策略（一般在前缀基础上）进行候选粗排并写到线上存储。
- 线上来一个请求 sub-query，直接查询存储获取召回结果，然后再基于训练的 pctr 预估模型或 pcpm 商业化导向进行重排，此时可以通过引入用户侧、context 侧等特征实现个性化排序。

7. 意图识别

搜索意图识别是 QU 最重要却也最具挑战的模块，存在的难点主要有：

- 用户输入 query 不规范：由于用户先验知识的差异，必然导致用户在通过自然语言组织表达同一需求时千差万别，甚至可能会出现 query 表达错误、遗漏等情况；
- 歧义性&多样性：用户的搜索 query 表达不够明确带来的意图歧义性或用户本身搜索意图存在多样性，比如：搜索 query "择天记"可能是想下载仙侠玄幻类游戏，可能是玄幻小说类 app，也可能是想看择天记电视剧而下视频类 app。此时衍生出来的另一个问题是当某个 query 存在多个意图可能时，需要合理地量化各个意图的需求强度；
- 如何根据用户及其所处 context 的不同实现个性化意图，比如用户的性别、年龄不同，搜索同一 query 的意图可能不一样，用户当前时刻搜索 query 的意图可能和上一时刻搜索 query 相关等。

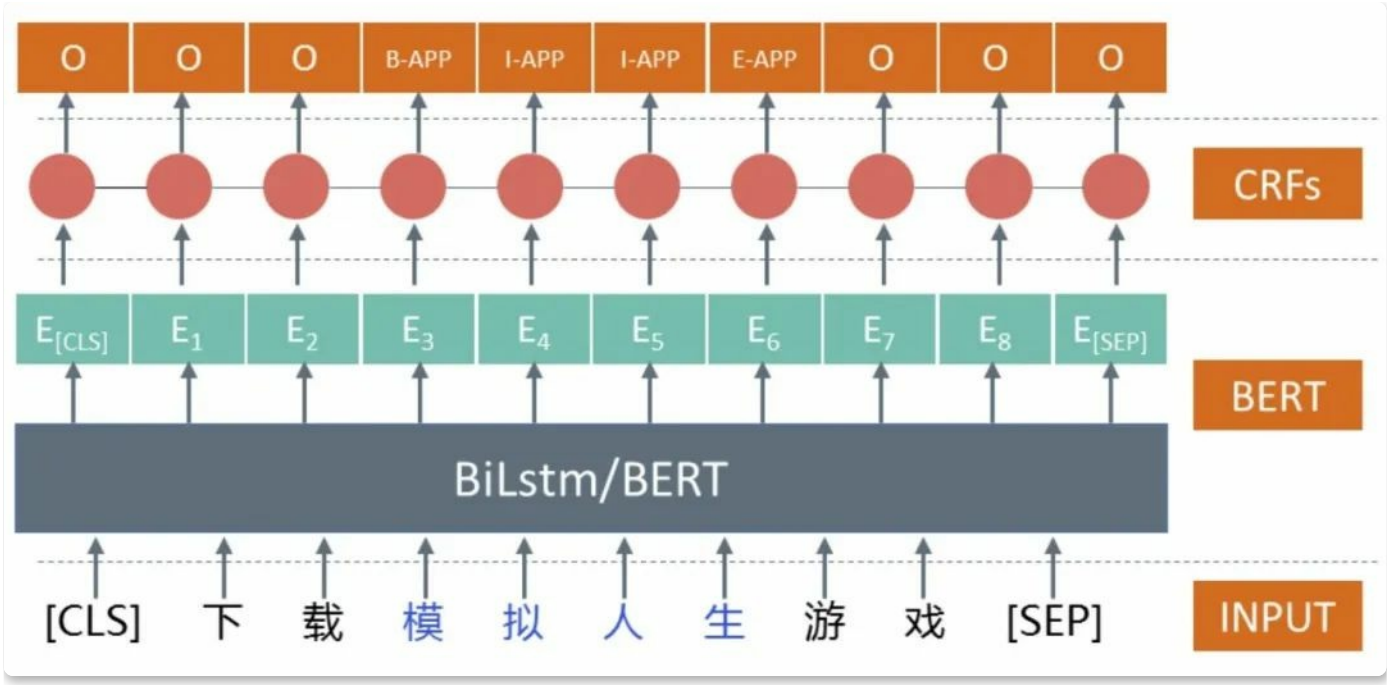
根据用户意图明确程度的差别，搜索意图识别又可以细分为精准意图和模糊意图识别。

① 精准意图

所谓精准意图，是指用户通过 query 所表达的意图已经非常明确了，其需求可以比较置信地锁定为一个资源目标。精准意图需求在垂直搜索中尤为常见，以应用市场 app 搜索为例，用户搜索 query "下载王者荣耀"很明确就是想下载"王者荣耀" app，这时候将"王者荣耀"展现在结果列表首位才合理。当然一般排序模型拟合足够好的情况下也能将对应的精准资源排在首位，但是以下一些情况可能会引起排序不稳定进而导致对应精准资源没能置顶在首位的问题：

- 长尾 query 行为特征稀疏，模型学习不够充分；
- 引入用户个性化特征，排序结果因人而异；
- 以商业化为导向的排序影响相关性体验。为此，需要一定策略识别出精准首位意图并将它们高优置顶，同时可以通过直达区产品形态给用户快速直达需求的搜索体验。

对于垂直搜索来说，精准意图一般是给定一个 query，找到与其意图精准对应的 item，可以通过文本匹配和 top 后验转化筛选出候选 item，然后通过从文本匹配、行为反馈、语义相似等方向构造样本特征训练 GBDT 等模型对 <query,item> 样本 pair 进行是否精准二分类。也可以尝试类似 DSSM 的语义匹配网络对 query 和 item 进行语义匹配。对于长尾 query 且完全文本包含 item 的情况，由于行为量不够丰富利用分类模型可能无法召回且直接进行文本匹配提取可能存在歧义性，此时可以视为 NER 任务通过训练 BiLSTM-CRF、BERT-CRF 等序列标注模型进行 item 实体的识别，再结合一些启发性策略及后验行为进行验证。



在 Google、百度等通用搜索中，用户可能会输入一些知识问答型的 query，此时用户的意图也比较明确，就是问题对应的答案。如搜索 query "刘德华的妻子是谁"，通过召回带"刘德华"、"妻子"字样的网页，用户估计也能找到答案，但如果能直接给出这个 query 的答案的话体验会更好，如下面百度搜索给出的结果。这时候可以归为 QA 问答任务来处理，一般需要结合知识图谱来做，也即 KBQA (Knowledge Based Question Answer)。

传统的 KBQA 做法是先对 query 进行词法句法以及语义解析，识别出其中的主要实体概念，再基于这些主题概念构造相应的查询逻辑表达式去知识库中进行查询及推理得到想要的答案。之后陆续有提出将问题和知识库中候选答案映射成分布式向量进行匹配，以及利用 CNN、LSTM、记忆网络等模型对应问题及候选答案向量建模优化等方法来进行 KBQA。

2. 模糊意图

模糊意图就是指用户的搜索意图不会具体到某个目标资源，而是代表一类或多类意图，比如用户搜索"视频app"，此时没有明确想下某款 app，可将其泛化到"视频类" tag 意图。模糊意图识别一般可以采用基于模板规则、行为统计反馈、分类模型等方法，这里主要会从意图分类及槽位解析两个方向进行阐述。

意图分类：

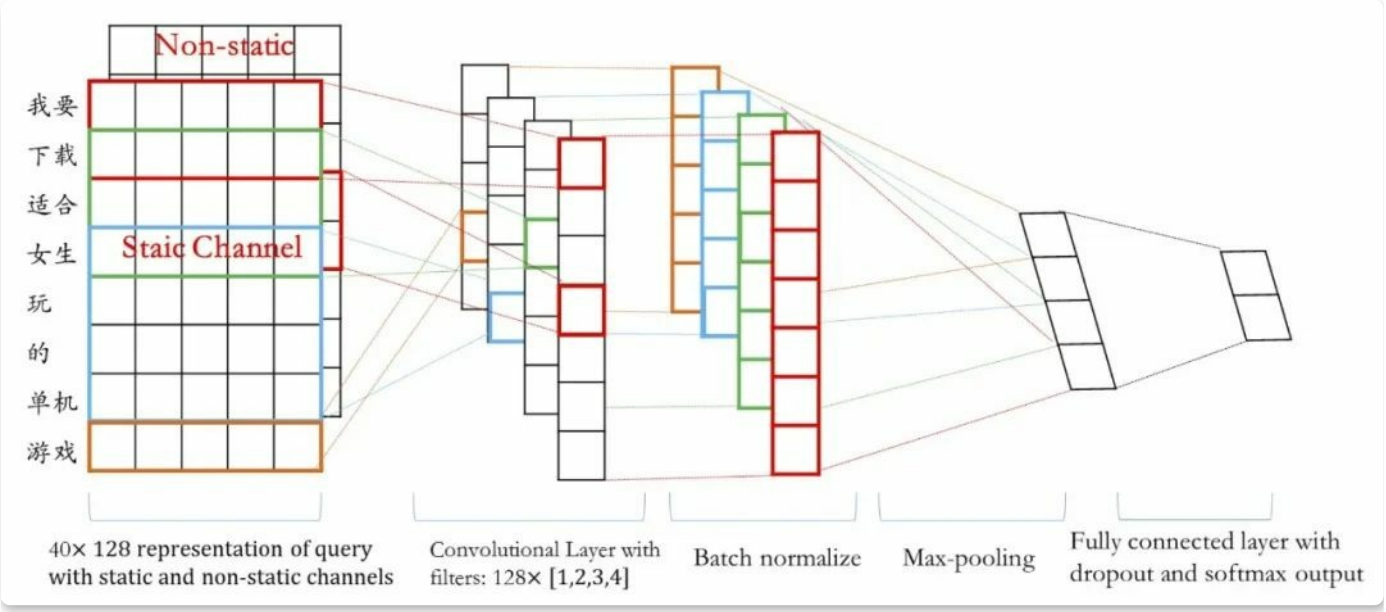
在构建 query 意图分类模型之前，需要先制定一套意图标签体系用于全面覆盖用户意图需求。这里需要对 query 侧和 item 侧的标签体系进行统一以便于在预测出某个 query 的意图 tag 分布后直接用 tag 去倒排索引中召回属于这些 tag 的 item。由于搜索 query 一般相对来说长度较短且意图存在多样性，意图分类可以归结为短文本多标签分类任务。

在意图分类样本构造方面，可以通过关联用户搜索点击行为分布及进行 item 理解获得的 tag 或站点所属行业分类信息等自动构造样本，对于可能存在的样本类别不平衡的问题，需要对样本进行重降采样等处理，或采用主动学习等方法进行高效的样本标注。至于模型方面，传统的文本分类主要采用向量空间模型 VSM 或进行其他特征工程来表征文本，然后用贝叶斯、SVM、最大熵等机器学习模型进行训练预测。

随着 Word2vec、GloVe、Fasttext 等分布式词向量技术的兴起，打破了传统 NLP 任务需要做大量特征工程的局面，通过分布式向量对文本进行表示后再接入其它网络结构进行端到端分类训练的做法成为了主流。如采用比较简单又实用的浅层网络模型 Fasttext，Fasttext 是从 Word2vec 衍生出来的，架构和 Word2vec 的类似，核心思想是将整篇文档的词及 n-gram 向量叠加平均得到文档向量，然后使用文档向量做 softmax 多分类。

相对 Word2vec 的优点是输入层考虑了字符级 ngram 特征可以一定程度解决 oov 问题以及输出层支持进行有监督的任务学习。使用 Fasttext 训练简单，且线上 inference 性能也很高，但也正因为采用相对简单的浅层网络结构其准确率也相对较低。为此，进一步地可以尝试一些深度神经网络模型，如：TextRNN、TextCNN、Char-CNN、BiLSTM+Self-Attention、RCNN、C-LSTM、HAN、EntNet、DMN 等。这些模型通过 CNN/RNN 网络结构提炼更高阶的上下文语义特征以及引入注意力机制、记忆存储机制等可以有效地优化模型的分类准确率。其实，对于 query 短文本分类来说采用相对简单的 TextRNN/TextCNN 网络结构就已经能达到较高的准确率了，其中 TextRNN 通过使用 GRU/LSTM 编码单元能更好地捕获词序和较长长度的上下文依赖信息，但由于采用 RNN 网络训练耗时相对较长。

TextCNN 则主要通过不同 size 的卷积核捕获不同局部窗口内的 ngram 组合特征，然后一般通过 max-pooling 或 kmax-pooling 保留变长文本中一个或多个位置的最强特征转换为固定长度的向量再做 sigmoid/softmax 分类。同时为进一步提升网络性能及加速模型收敛，还可以在网络中进一步考虑 dropout 及 batch normalize 等防过拟合机制，以及考虑在输入层融入 Word2vec、GloVe、Fastext 等模型预训练得到的 embedding，如下图在 cnn 输入层中加入预训练 embedding 组成双通道输入。虽然 TextCNN 对于捕获长程依赖信息方面会不如 TextRNN，考虑到 query 一般长度相对较短所以影响相对还好，而且其训练速度及在线 inference 性能也都比较符合要求。

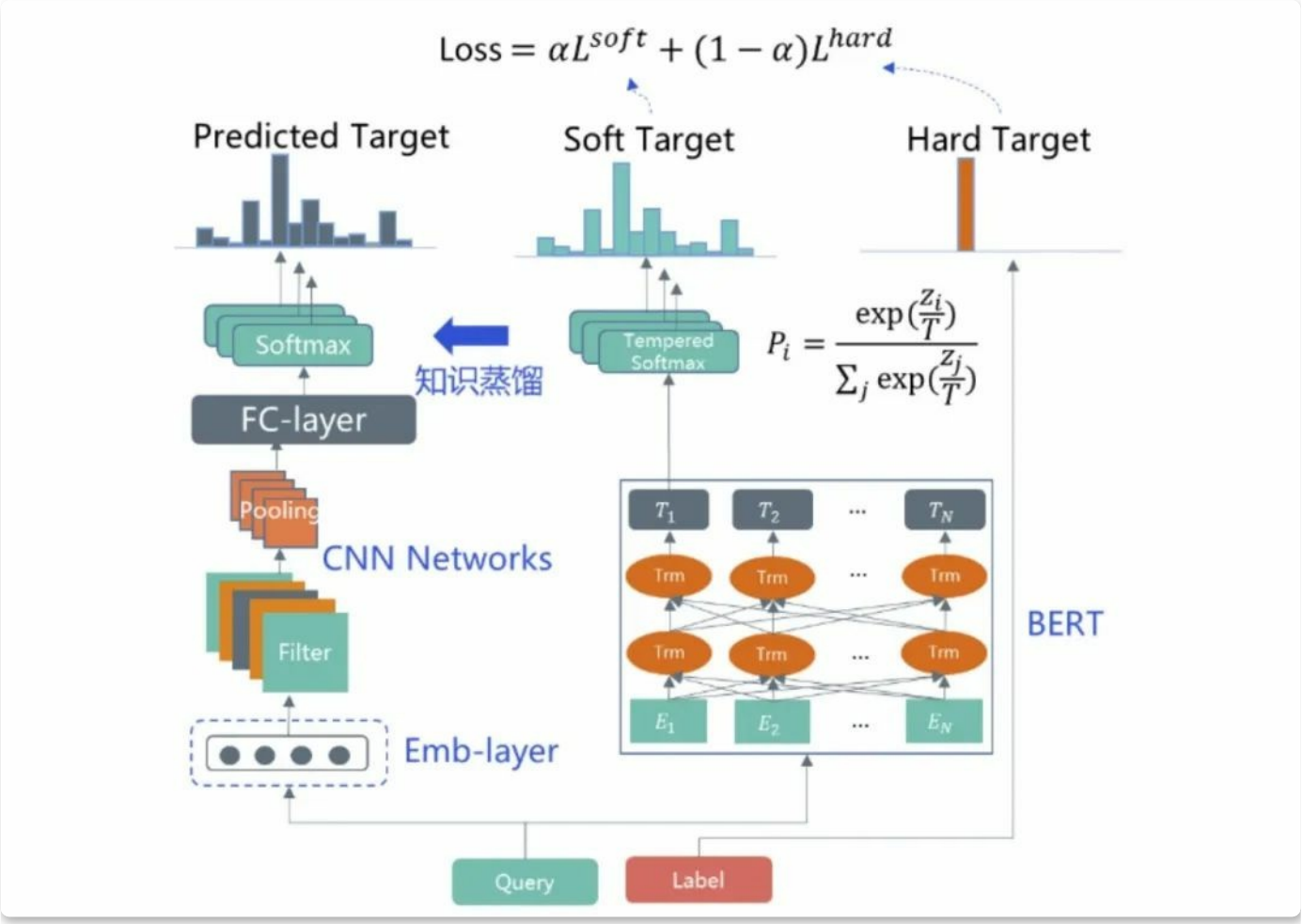


除了从零开始训练或引入无监督预训练的隐式 embedding 表示，还可以通过引入显式的知识概念进一步丰富文本的语义表达，在有比较丰富的领域知识库的情况下进行 NER 实体识别，然后在模型的输入中可以融入这些实体知识特征，通过引入外部知识来优化分类的模型有 KPCNN、STCKA 等。由于 Word2vec、GloVe 等模型训练得到的词语 embedding 对不同的上下文来说都是固定

的，无法解决一词多义等问题，基于此陆续提出的 ELMO、GPT、BERT 等深度预训练语言模型渐渐成为了 NLP 任务的标配，这些模型及其各种演进版本在多个 GLUE 基准中均取得了进一步的突破。

通过在大规模语料上预训练得到的 BERT 等模型能较好地动态捕获词语在不同上下文中的前后向语义表达，基于这些预训练模型在意图分类任务上进行 finetune 学习能进一步提升模型分类准确率。同样地，像 ERNIE 等模型通过引入外部知识也能进一步提升模型效果，尤其在一些垂直搜索有较多特定领域实体的情况下，可以尝试将这些领域实体知识融入模型预训练及 finetune 过程中。

由于 BERT 等模型复杂度较高，进行在线 inference 时耗时也相对较高可能达不到性能要求，为此需要在模型精度和复杂度上做个权衡，从模型剪枝、半精度量化、知识蒸馏等方向进行性能优化。这里可以尝试通过权重分解及参数共享等方法精简优化的 ALBERT 模型，也可以尝试诸如 DistilBERT、TinyBERT 等知识蒸馏学习模型。



槽位解析：

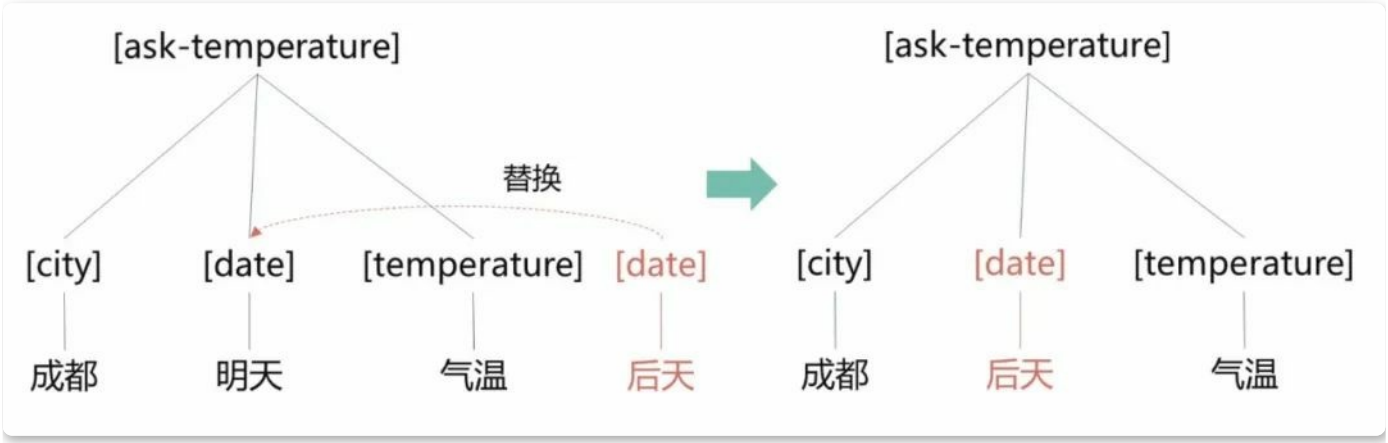
前面介绍的各种基于深度学习模型的意图分类能起到比较好相关性导航的作用，如将 query 意图划分到"天气"、"酒店"、"汽车"等意图体系中。但是针对更加复杂的口语化 query，我们需要进行识别提取出 query 中重要的意图成分以进行更全面的意图理解，此时仅进行意图分类是不够的。

比如对于搜索 query "北京飞成都的机票"，意图分类模型可以识别出是"订机票"的意图，但是无法区分出 query 中的出发地和目的地信息，需要通过一定方法识别出"出发地"概念及其对应值是"北京"、"目的地"概念及其对应值是"成都"，基于此可以作出一些决策提供更直观的结果以提升用户搜索体验。

由于自然语言表达充满着歧义性，计算机肯定是无法直接理解的，需要将 query 表示成计算机能够理解的表示。类似于计算机语言无法理解高级编程语言一样，需要通过将高级编程语言代码编译成低级的汇编或二进制代码后计算机才能执行。所以我们也需要一个类似的"编译器"能将 query 按一定文法规则进行确定性的形式化表示，可以将这个过程称之为语义解析，前面提到的传统 KBQA 的做法也需要该技术将 query 转换成相应的形式化表示才能进一步执行推理等操作。

对于 query "北京飞成都的机票"，通过意图分类模型可以识别出 query 的整体意图是订机票，在此基础上进一步语义解析出对应的出发地 Depart="北京"，到达地 Arrive="成都"，所以生成的形式化表达可以是：Ticket=Order(Depart,Arrive)，Depart={北京}，Arrive={成都}。如果 query 换成是"成都飞北京的机票"，同样的需要解析出 query 的意图是订机票，但是出发地和到达地互换了，所以语义解析过程需要除了需要对概念进行标注识别，还可以通过对概念进行归一来提高泛化性，比如这里"北京"、"成都"都可以归一为 [city] 概念，形式化表达变为：Ticket=Order(Depart,Arrive)，Depart={city}，Arrive={city}，其中 city={北京、上海、成都…}。形式化表达可以递归地定义为一些子表达式的组合的形式，为进行语义解析，最主要的是确定一种形式化语言及对应的解析器，通常我们采用确定的上下文无关文法以确保形式化的每一部分都对应有一个解析树。

得到 query 对应的形式化表示后，还可以进行一些解析树归并等形式化运算推演。为此，除了可以用来理解搜索 query 的结构化语义，语义解析技术也广泛应用于 QA 问答及聊天机器人中，如多轮对话中比较有挑战的上下文省略和指代消歧问题也可以一定程度通过将下文 query 对应的解析树合并变换到上文 query 对应的解析树中来解决。如下图例子所示，当用户在第一轮对话询问"成都明天气温？"时构造出相应的解析树，接着问"后天如何？"时就可以将日期进行替换后合并到之前的解析树中。

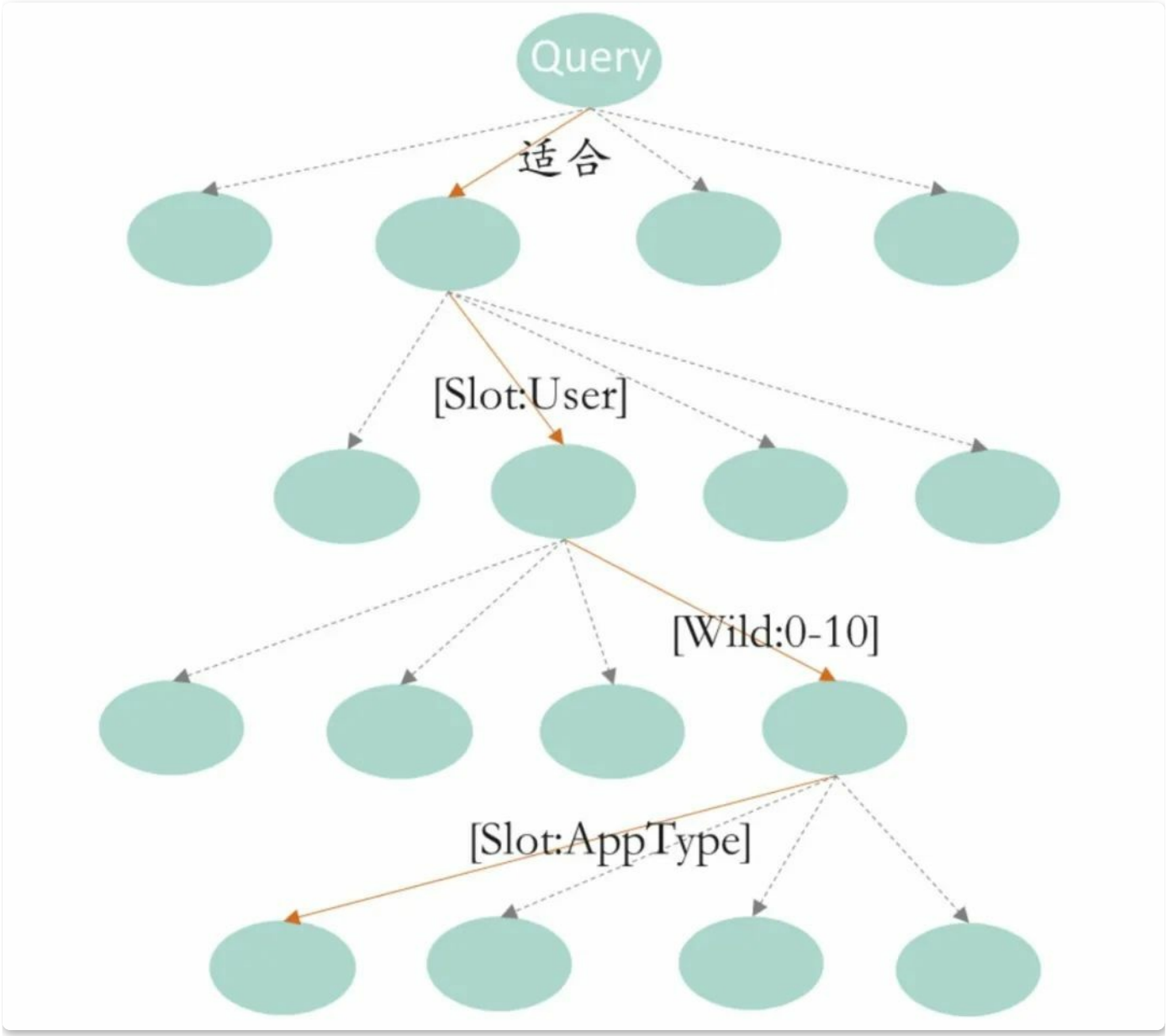


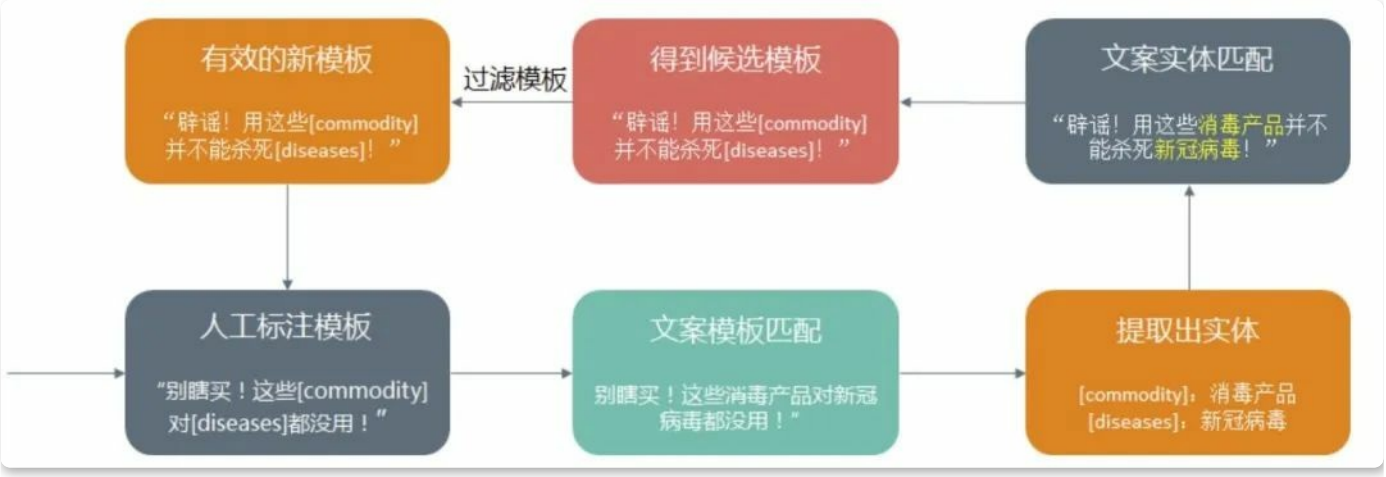
目前学术界和工业界在形式化语言和语义解析器方面均有一定的研究成果，通过一些基于统计、半监督、监督的方法来训练得到语义解析器，其中比较有名的开源语义解析器有 Google 的 SLING、SyntaxNet。简单的做法是通过人工制定的正则表达式和槽位解析的方法来进行语义解析，正则表达式相对好理解，槽位解析是指通过将具有相同模式的 query 归纳成模板，基于模板规则来解析用户 query 意图及意图槽位值。模板构成主要包括：槽位词、固定词、函数、通配符，其形式化表达变为中槽位词是指能抽象到某一概念的词语集合，如："北京"、"上海"、"成都"这些词都可以抽象到城市概念，固定词也即明确的某个词语，可以通过函数来匹配一

些诸如数字组合等词，通配符可以匹配任意字符。

我们需要结合领域的业务知识来构造模板，构造过程需要保证模板符合语法句式规范，同时尽量保证其泛化性以覆盖更多的query问法。举个简单例子，构造模板："适合[Slot:User][Wild:0-10][Slot:AppType]"及相应的槽位词　[Slot:User]={女生，男生，小孩，老人，…}、[Slot:AppType]={单机游戏，益智游戏，moba游戏，…}，然后通过构建如下图所示的相应 trie 树，可以自上而下遍历及回溯解析出 query "适合女生玩的单机游戏"匹配上了这个模板，从而识别出 query 整体意图是询问女生类单机游戏，其中用户槽位值为"女生"，app类型槽位值为"单机游戏"，"适合"是固定词，"玩的"匹配通配符　[Wild:0-10]，基于这些槽位解析的结果，接下来可以进行一系列的决策。槽位解析方法的优点是准确率高，可控性强，解释性好，但缺点是需要耗费较多地人力进行对模板及槽位词等进行标注，同时维护起来也比较麻烦。

可以考虑结合一些策略方法来一定程度减少人工标注量，如基于前面提到的同义词挖掘技术及词向量化挖掘同位/上下位词等方法来辅助槽位词的标注，以及在人工进行模板标注的基础上采用 bootstrapping 迭代挖掘构造出更多的模板。槽位解析的具体实现可以参考的开源项目 Snips-nlu，进行意图识别的同时从自然语言句子中解析提取结构化槽位信息。





8. 敏感识别

敏感识别模块主要对 query 进行是否带有色情、反动、赌博、暴力等敏感话题的识别，如果识别出 query 中存在敏感话题可以进行定向改写到相对合适的 query 或者给用户做搜索引导提示等处理。敏感识别可以归为分类问题，最简单的做法就是词表匹配，按不同的敏感话题人工输入一批词库，复杂点就训练一个分类模型进行多分类，传统的 SVM、最大熵分类或者 TextCNN、TextRNN 等模型都能比较好地 work。

9. 时效性分析

用户的搜索需求可能会显式或隐式地带有一定的时效性要求，如："最近上映的好看电影"带有显式的时间词"最近"，而"疫情进展"则隐式地表达了解最新情况的需求。时效性大概可以分为三种：持续时效性，周期时效性，准/实时时效性。其中持续时效性是指 query 一直具有时效性，如："美食推荐"，周期时效性是指具有周期性、季节性的事件或需求，如："世界杯"、用户在冬季搜索"上衣"等，而准/实时效性是指近期发生或突然发生的事件。

通过分析出 query 的时效性需求等级的不同，在召回 item 时就可以针对性地做一些过滤或者在排序时进行时效性调权。其中显式的时效性需求因为带有时间关键词，可以通过规则匹配或训练分类模型进行判断识别，而隐式表达的时效性则可以通过按时间维度分析历史搜索 qv 行为、实时监测最新搜索 qv 变化情况以及综合考虑搜索意图及当前时间上下文等方法来判断识别。

结语

人工智能的发展是循序渐进的，需要经历从计算智能到记忆智能、感知智能、认知智能，最终到达创造智能等多个发展阶段才能称得上是真正的人工智能。目前业界在计算、记忆和感知方面已经做得相对比较成熟，但是在认知智能方面则还需要有进一步突破，而 NLP 恰好是实现认知智能最重要的一环，为此 NLP 也被称为人工智能皇冠上的一颗明珠。同样地，真正的语义搜索也远未到来，对 query 的语义理解能力很大程度决定着整个搜索的智能程度。

本文仅为个人在进行query理解相关优化项目的一些简单总结，主要从搜索的角度对 query 理解涉及的各个重要模块的概念及其对应的一些方法进行阐述。文中暂未涉及太深的技术探讨，希望能帮助到大家对搜索 QU 相关概念有一个初步的认识，起到抛砖引玉的效果，如有错误及不足之处，还请不吝交流指出。

- 算法工程师的效率神器——vim篇
- 硬核推导Google AdaFactor：一个省显存的宝藏优化器
- 卖萌屋上线Arxiv论文速刷神器，直达学术最前沿！

夕小瑶的卖萌屋

关注&星标小夕，带你解锁AI秘籍

订阅号主页下方「撩一下」有惊喜哦

参考文献

[1] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

[2] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. Acl, 655–665.

[3] Armand Joulin, Edouard Grave , Piotr Bojanowski, omas Mikolov. Bag of Tricks for Efficient Text Classification. arXiv, 2016.

[4] Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio. Neural Machine Translation By Jointly Learning To Align And Translate. ICLR, 2015.

[5] Sanjeev Arora, Yingyu Liang, Tengyu Ma. A Simple but Tough-to-Beat Baseline for Sentence Embeddings,2016.

[6] C Zhou, C Sun, Z Liu, F Lau. A C-LSTM neural network for text classification. arXiv, 2015.

[7] X Zhang, J Zhao, Y LeCun. Character-level convolutional networks for text classification. NIPS, 2015.

[8] S Lai, L Xu, K Liu, J Zhao. Recurrent convolutional neural networks for text classification. AAAI, 2015.

[9] Z Lin, M Feng, CN Santos, M Yu, B Xiang. A structured self-attentive sentence embedding. arXiv, 2017.

[10] J Howard, S Ruder. Universal language model fine-tuning for text classification. arXiv, 2018.

[11] J Devlin, MW Chang, K Lee, K Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv, 2018.

[12] J Wang, Z Wang, D Zhang, J Yan. Combining Knowledge with Deep Convolutional Neural Networks for Short Text Classification. IJCAI, 2017.

[13] J Chen, Y Hu, J Liu, Y Xiao, H Jiang. Deep short text classification with knowledge powered attention. AAAI,2019.

[14] H Ren, L Yang, E Xun. A sequence to sequence learning for Chinese grammatical error correction. NLPCC,2018.

[15] Y Hong, X Yu, N He, N Liu, J Liu. FASPell: A Fast, Adaptable, Simple, Powerful Chinese Spell Checker Based On DAE-Decoder Paradigm. EMNLP, 2019.

[16] I Antonellis, H Garcia-Molina. Simrank++ query rewriting through link analysis of the clickgraph. WWW'08.

[17] G Grigonytė, J Cordeiro, G Dias, R Moraliyski. Paraphrase alignment for synonym evidence discovery. COLING,2010.

[18] X Wei, F Peng, H Tseng, Y Lu, B Dumoulin. Context sensitive synonym discovery for web search queries. CIKM,2009.

[19] S Zhao, H Wang, T Liu. Paraphrasing with search engine query logs. COLING, 2010.

[20] X Ma, X Luo, S Huang, Y Guo. Multi-Distribution Characteristics Based Chinese Entity Synonym Extraction from The Web. IJISA, 2019.

[21] H Fei, S Tan, P Li. Hierarchical multi-task word embedding learning for synonym prediction. KDD,2019.

[22] M Qu, X Ren, J Han. Automatic synonym discovery with knowledge bases. KDD,2017.

[23] J Shen, R Lyu, X Ren, M Vanni, B Sadler. Mining Entity Synonyms with Efficient Neural Set Generation. AAAI,2019.

[24] A Vaswani, N Shazeer, N Parmar. Attention is all you need. NIPS, 2017.

[25] Berant J, Chou A, Frostig R, et al. Semantic Parsing on Freebase from Question-Answer Pairs. EMNLP,2013.

[26] Cai Q, Yates A. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. ACL,2013.

[27] Bordes A, Chopra S, Weston J. Question answering with subgraph embeddings. arXiv,2014.

[28] Dong L, Wei F, Zhou M, et al. Question Answering over Freebase with Multi-Column Convolutional Neural Networks. ACL,2015.

[29] E Malmi, S Krause, S Rothe, D Mirylenka. Encode, Tag, Realize: High-Precision Text Editing. arXiv, 2019.