Query 理解和语义召回在知乎搜索中的应用

原创 方宽 DataFunTalk 2020-01-02

分享嘉宾：方宽 知乎 算法工程师

文章整理：艺饭饭

内容来源：DataFunTalk

出品平台：DataFun

注：欢迎转载，转载请留言。

导读：随着用户规模和产品的发展， 知乎搜索面临着越来越大的 query 长尾化挑战，query 理解是提升搜索召回质量的关键。本次分享将介绍知乎搜索在 query term weighting，同义词扩展，query 改写，以及语义召回等方向上的实践方法和落地情况。

知乎搜索的历史

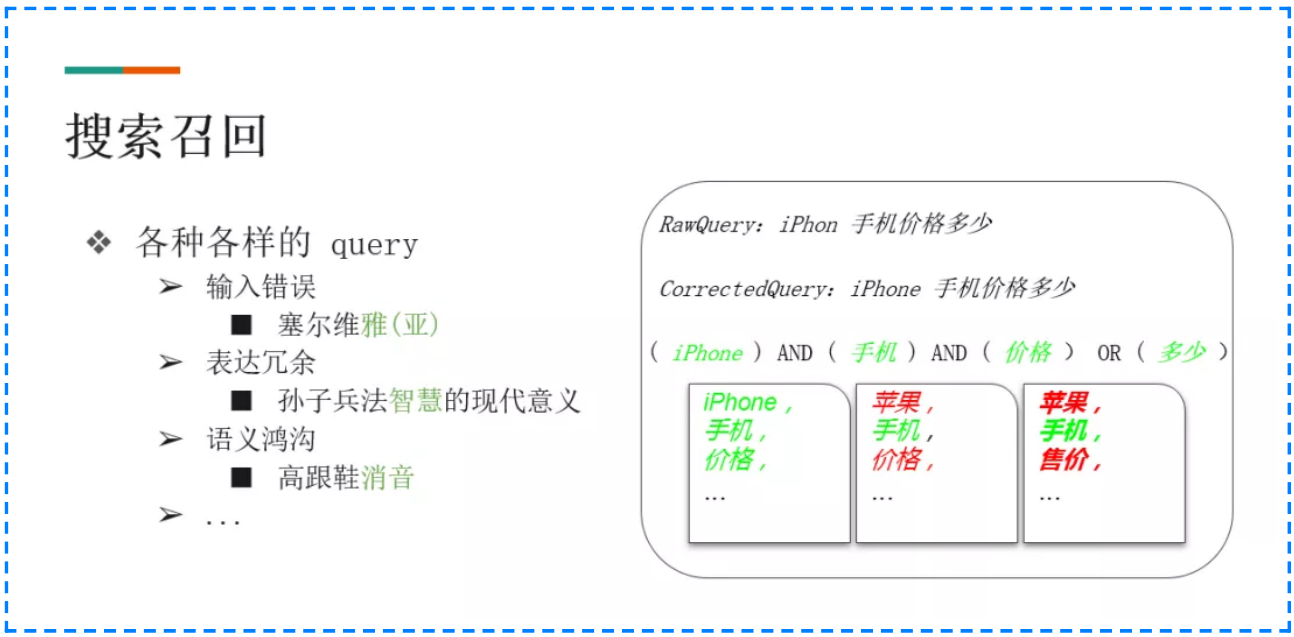
首先回顾下知乎搜索的历史。现有的知乎搜索16年开始内部研发，17年中完全替换外部支持的系统。算法团队从 18 年初开始成立，规模5人左右，之后开始了快速的迭代过程。

主要进展有: 18年 4 月份上线 Term Weight；6 月份上线自研的 Rust 引擎并已对外开源；后续上线一系列的算法模型如纠错、语义相关性，以及今天会重点分享的语义索引和 query 改写；今年下半年工作主要集中在排序侧，主要为用 DNN 替换了树排序模型，并在此基础上继续上线了若干项收益不错的优化。由于今天分享的内容主要是在 NLP 范围，排序侧的内容可以后续有机会再分享。

知乎是一个问答类型的社区平台，内容以用户生产的问题和回答为主。问答网站的属性导致了知乎搜索 query 词的长尾化。实际上在大搜领域中问答类 query 也是相对满意度比较差的一类，而目前各类垂搜分流了原来大搜的流量，当中知乎搜索承担了一部分相对比较难处理的 query。

从我们内部实际数据来看，近一年知乎搜索的长尾化趋势也越来越明显。如上图右上部分是头腰尾部搜索比例，可以看到长尾 query 增长趋势明显。但在我们搜索团队的努力下，用户的搜索满意度并没有随着 query 难度的增加而下降。

长尾 query 特点



长尾 query 的多样性对于搜索系统来说是一个很大的挑战，原因有：

❶ 存在输入错误。例如上图中的错误 query "塞尔维雅"（塞尔维亚），对于这种错误我们希望系统能够自动的纠错；

❷ 存在表达冗余。例如输入 "孙子兵法智慧的现代意义"，在这个语境下，"智慧" 是一个无关紧要的词。如果强制去匹配 "智慧" 的话，反而匹配不出真正想要的结果；

❸ 存在语义鸿沟。比如 "高跟鞋消音"，其中 "消音" 这个词的表达较少见，使得同时包含 "高跟鞋" 和 "消音" 文档较少。而类似的表达如 "高跟鞋声音大如何消除"、"高跟鞋消声" 等可能较多。用户输入的 query 和用户生产内容之间存在了语义鸿沟。其他类型的难点还有表达不完整，意图不明等等。

我们先通过图右边这个query："iPhone 手机价格多少" 来介绍如何解决上述问题：

❶ 对于输入错误，比如说用户输入的 query 是 iPhone，但是这个词输错了，会通过纠错模块将其纠正为正确的 query；

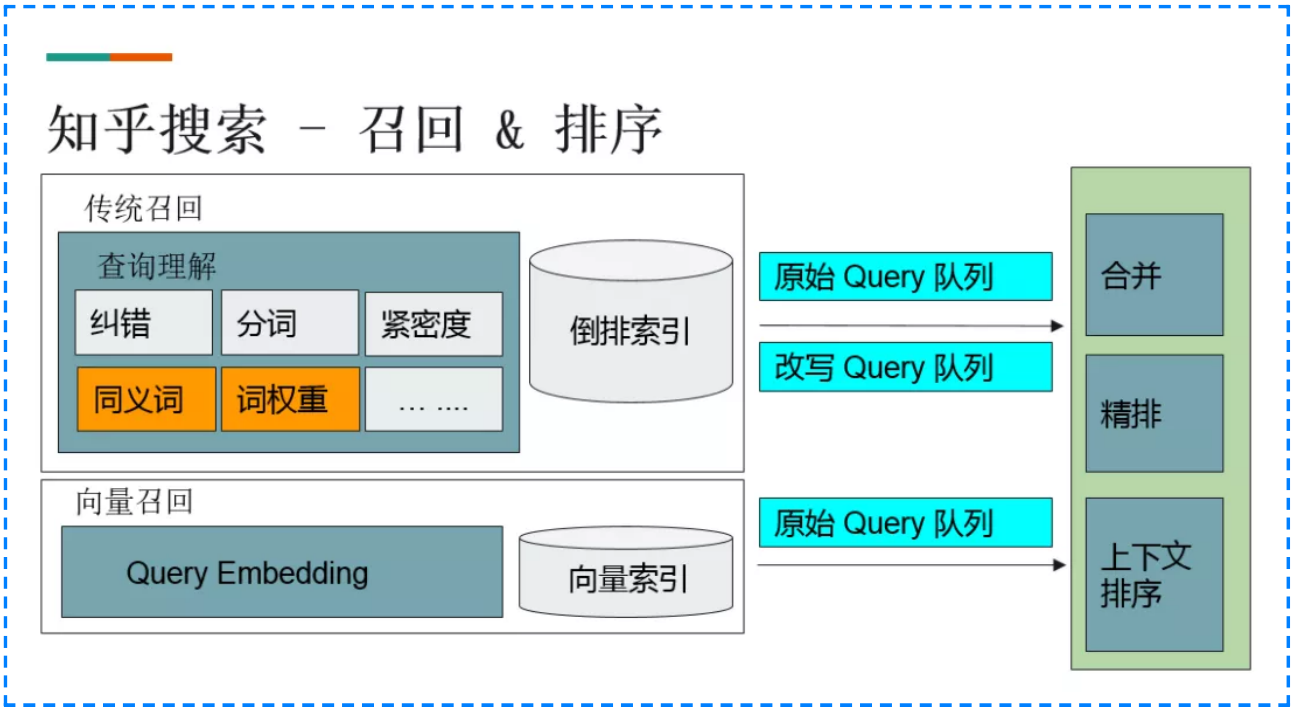
❷ 对于表达冗余，通过计算每一个词的重要程度也就是 term weight，来确定参与倒排索引求交操作的词。先对 query 进行分词，切成 iPhone、手机、价格、多少，之后判断各词对于表达意图更重要，重要的词会在检索时参与倒排索引的求交操作，不那么重要的词不严格要求一定在文档中出现；

❸ 解决语义鸿沟的问题。需要对原始 query 做同义词扩展，比如 "iPhone" 和 "苹果" 是同义词，"价格" 和 "售价" 是同义词。

所以在传统的搜索领域中的查询模块，往往包含这些子任务：纠错、分词、紧密度、同义词、词权重，以及其他的如实体词识别、意图识别等等。这些查询理解会生成一个结构化的数据，

检索模块就可以通过结构化的查询串去召回相关的文档。

知乎搜索的召回和排序



知乎的召回系统主要分为两部分：

❶ 基于词的传统的倒排索引召回；

❷ 基于向量的向量索引的召回机制。

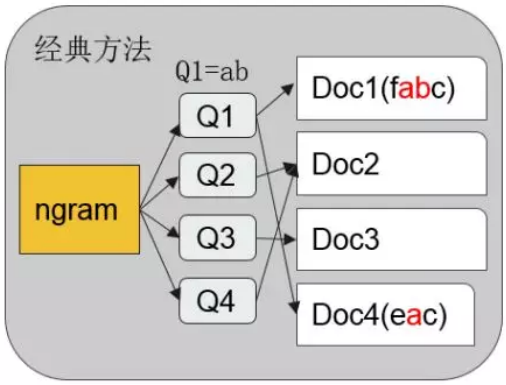
如图所示，实际召回中有三个队列，基于倒排索引的队列占两个，分别是原始 Query 队列和改写 Query 队列。改写队列是用翻译模型去生成一个表达更适合检索的 query，然后再去做 query 理解和索引召回。而第三个队列是基于 embedding 的索引。首先把所有 doc、原始 query 都转为向量，再通过空间最近邻的 KNN-Search，找到与 query 最相似的文档。

最终参与排序的就是这三个队列，并进行合并和精排，在精排之后会进行上下文排序。下面来介绍下各个模块具体的实现方法。

Query Term Weight

Query Term Weight

- IDF 词典
- 语境动态调整：
  $TermRecall = P(t|R) = 1 - TermMismatchRate$ [1]
- 训练数据
  - 点击日志（弱监督）
  - 标注数据
- 模型
  - Embedding based [2]



1.Zhao L, Callan J. Term necessity prediction[C]//Proceedings of the 19th ACM international conference on Information and knowledge management. ACM, 2010: 259-268.
2.Zheng G, Callan J. Learning to reweight terms with distributed representations[C]//Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. ACM, 2015: 575-584.

对于 term weight，最简单的实现方式是用 IDF 逆文档频率算出，即看这个词在语料里的出现次数，如果次数较少，则认为其包含的信息量是相对较多。

这个方法的局限是无法根据 query 上下文语境动态适应，因为其在不同的上下文中 weight 是一致的。所以我们再通过统计点击数据中进行调整。如图，如果用户的 query 包含 a，b 两个 term，并且点击了 Doc1 和 Doc4，其中 Doc1 包含 a，b， Doc4 只包含a，即 a 出现2次，b 出现1次。一个朴素的想法就是 a 的权重就是1，b 是0.5，这样就可以从历史日志里把每个 query 里的 term 权重统计出来。

这个方法对于从未出现过的 query 就无法从历史数据中统计得到。所以我们从 query 粒度泛化到 gram 粒度，按 ngram 来聚合。比如 a，b 就是一个 bigram，类似于 mapreduce 的 map 过程，首先会把 query 中的 ngram 先 map 出来，再发送到 reduce 里面统计。这样就能统计出每个 ngram 下较置信的权重。

对于 ngram，在实际操作中需要注意以下几点：

❶ 词表较大，往往在十亿量级以上；

❷ 字典更新较慢；

❸ 无法处理长距离依赖。基本最多到 3gram，多 gram 词不仅会使词表更加膨胀，词与词的依赖关系也将更难捕捉到；

❹ 无法解决过于稀疏的词，过于长尾的词会退化成 idf。

我们的解法是把 ngram 基于词典的泛化方法变为基于 embedding 的模型泛化方法。目标是想得到根据 query 语境动态自适应的 term weight。如果可以获取跟 query 上下文相关的动态词向量，那么在词向量的基础上再接 MLP 就可以预测出词权重了。目前的方法是通过 term 词本身的词向量减去 query 所有词 pooling 之后的词向量获得，今后也会尝试替换成 ELMo 或者 Transformer 的结构以获得更好的效果。

同义词扩展

对于同义词扩展策略，首先需要一个同义词词表。现在有可以直接应用的公开数据集。但是数据集的词与知乎搜索里的词可能由于分词粒度不一致而不匹配，并且同义词无法保证更新。所以我们主要用内部语料进行数据挖掘，主要方向为：

❶ User logs: session

用户搜索日志，利用用户搜索一个 query 后改写了的新 query，把两者当作相关关系，用对齐的工具将两者同义词找出；

❷ Query logs: click

类似的，利用两个 query 点击的相同 doc 构建 query 关系；

❸ Doc：pretrain embedding

Doc 本身的语料。主要利用知乎内部数据去训练 word embedding 的模型，用 embedding 的值计算相关性。

同义词扩展

- Embedding：无法区分同义词和反义词；无法区分语义相似词和概念相关词
- Conter-fitting：使用 Antonym pair 和 Synonym pair 进行 finetune

loss 1 ：增大Antonym pair 距离

$$\mathrm{AR}(V') = \sum_{(u,w)\in A} \tau\left(\delta - d(\mathbf{v}'_u, \mathbf{v}'_w)\right)$$

loss 2：减小Synonym pair 距离

$$\mathrm{SA}(V') = \sum_{(u,w)\in S} \tau\left(d(\mathbf{v}'_u, \mathbf{v}'_w) - \gamma\right)$$

loss3：保留原始信息（正则项）

$$\mathrm{VSP}(V, V') = \sum_{i=1}^{N} \sum_{j\in N(i)} \tau\left(d(\mathbf{v}'_i, \mathbf{v}'_j) - d(\mathbf{v}_i, \mathbf{v}_j)\right)$$

	East	Expensive	British
Before	West, North, South, Southeast, northeast	Pricy, Cheaper, Costly, Overpriced, inexpensive	American, Australian, Britain, European, England
After	Eastward, Eastern, easterly	Costly, Pricy, Overpriced, Pricey, afford	Brits, London, BBC, UK, Britain

Mrkšić, Nikola, et al. Counter-fitting Word Vectors to Linguistic Constraints. arXiv preprint arXiv:1603.00892 (2016).

Word2vec 这一类的 embedding 计算方法，核心假设是拥有相似上下文的词相似。需要注意该方法无法区分同义词和反义词/同位词，例如最大/最小，苹果/香蕉。我们这里利用监督学习的信号，对训练完成的词向量空间进行微调。

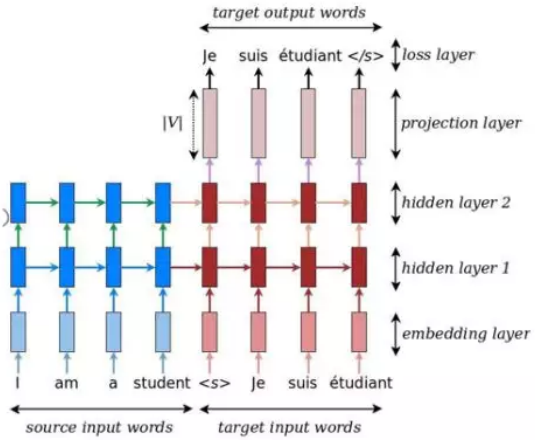
❹ 外部数据

用一些规则如百科里 "XXX 又名XXX" 句式进行挖掘。

Query 改写

Query 改写 - NMT

- 训练语料
  - Query to doc title
  - Doc to query
  - Query to query （co-click, session, simi_rank++）
- 模型
  - Google NMT
- 语料清洗
  - Language model （罕见->常见）
  - 相关性过滤
  - bpe 切词



He, Yunlong, et al. "Learning to rewrite queries." *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016.

接下来分享的是 query 改写部分。为什么要做 query 改写?

在传统搜索里，一个 query 进来需要做多个子任务的处理。假如每个任务只能做到90%，那么积累起来的损失会非常大。我们需要一个方法一次性完成多个任务去减小损失，比如把 "苹果手机价格多少" 直接改写为 "苹果手机售价"。

用翻译方法可以实现这个需求，即把改写问题看成 src 和 target 语言为同一种语言的翻译问题，翻译模型用的是谷歌的 NMT 结构。

对于这个任务来说，训练语料的挖掘比修改模型结构更为重要，我们通过用户行为日志来挖掘训练语料。比如从 query 到 title，点击同一 doc 的不同 query，都可形成平行语料。

最后我们用来训练模型的语料用的是 Query -> Query 的平行语料，因为两者在同一个语义空间，长度也基本接近。如果 query 翻译成 title，则会因为 title 长度通常过长而加大训练复杂度。

具体方法是：

❶ 语言模型的过滤

目的是把罕见的表达换为常见的表达。用 ngram 模型算一个得分，把较罕见的放在前面；

❷ 进行相关性过滤

考虑到点击日志噪声比较多，通过相关性过滤掉噪声；

❸ 确定切词粒度

用的是 BPE 的 Subword 方法。未用知乎内部的切词是因为未登录词无法预测，同时词表较大，训练速度较慢。

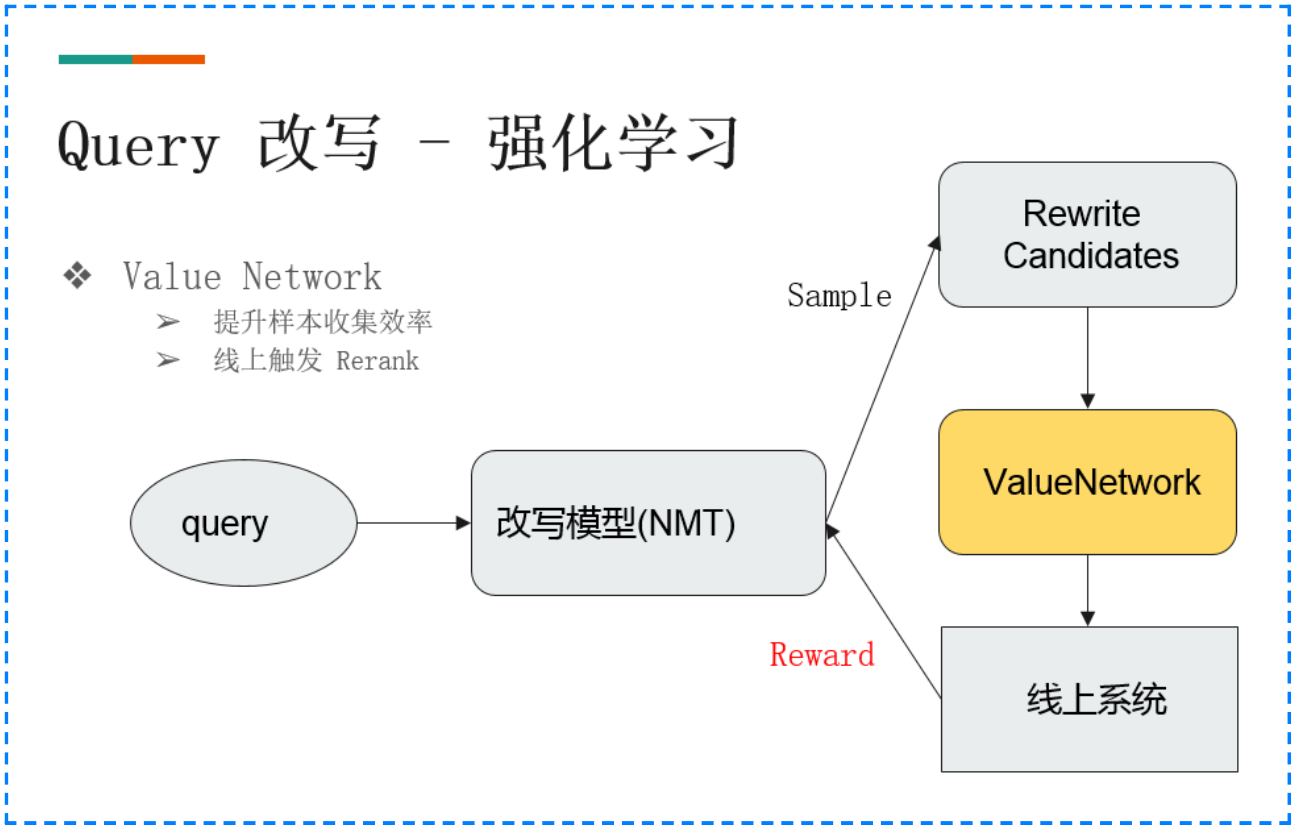
Query 改写 – 强化学习

具体来说，改写模型是拿共现数据做训练语料，但不能保证改写出来的结果一定可以召回有用的文档。我们参考了谷歌 2018 ICLR 的工作，为了将用户输入的 query 改写为更容易检索出答案的 query，把 QA 系统当做一个 environment，改写模型当做 agent，CNN 是对所有召回结果做选择的模块。

在我们的场景中，将图中灰色框里的 QA 对应一套倒排索引，CNN 对应在线排序服务模块。

首先会把原始 query 的召回结果和改写 query 的召回结果一并送入排序服务模块，然后得到 topK 的排序；之后统计这 topK 结果里有多少是改写 query 召回队列里的结果，将其作为 reward。

接下来，我们用强化学习 policy gradient 方法来 finetune 改写模型。利用强化学习的一个好处是，不再需要对齐文本语料，只需要随机的 query 即可。



在利用强化学习训练时遇到的问题之一是 reward 稀疏问题。我们发送了很多 NMT 模型的结果给线上系统，但在返回来的 topK 列表里并不包含改写队列的召回。这导致了训练的过程非常的缓慢，也对线上系统造成了很大的开销。

为了缓解 reward 稀疏问题，参照 alphago 里面的模式，即由策略网络，价值网络和蒙特卡洛树搜索三个组件完成。我们可以把 NMT 当成是一个策略网络，然后这里主要添加了价值网络。价值网络用来评估一次改写完成以后，其实际能从系统当中拿到多少 reward。他的输入是两个拼接在一起的 query，输出是一个浮点值。用强化学习的经验来学习这个价值网络，然后将其后面的训练过程中来加速训练。

最后的优化效果非常明显，在加入价值网络以后 reward 比例提升了50%以上。

Query 改写 - 上线效果

- 线上覆盖率

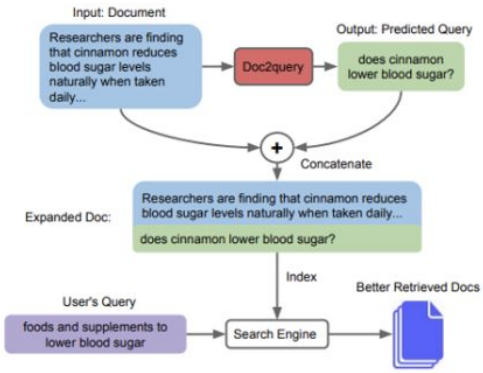
	Top20 包含 Rewrite 比例
Query 数量	24%
Query Doc 数量	4%

```
raw_query = [造型精油推荐]
rewrite_query = [护发精油推荐]
rewrite_query = [头发精油推荐]
rewrite_query = [头发精油]
rewrite_query = [美发精油]
```

同时，这个价值网络还可以用于在线改写触发 rerank。如下图所示，线上覆盖率有了明显提升， top20 包含 rewrite 比例的 query 数量占到了24%，query doc 数量占到了4%。

Query 改写 - Doc2Query

- 传统检索字段
  - Title / Body / Meta…
  - Clicked Query
- 问题
  - 覆盖率有限
  - 展现偏置
- Doc2Query
  - 训练语料
    - Doc -> Query
  - 替换索引 Clicked Query field



Nogueira, Rodrigo, et al. "Document Expansion by Query Prediction." *arXiv preprint arXiv:1904.08375* (2019).

APA

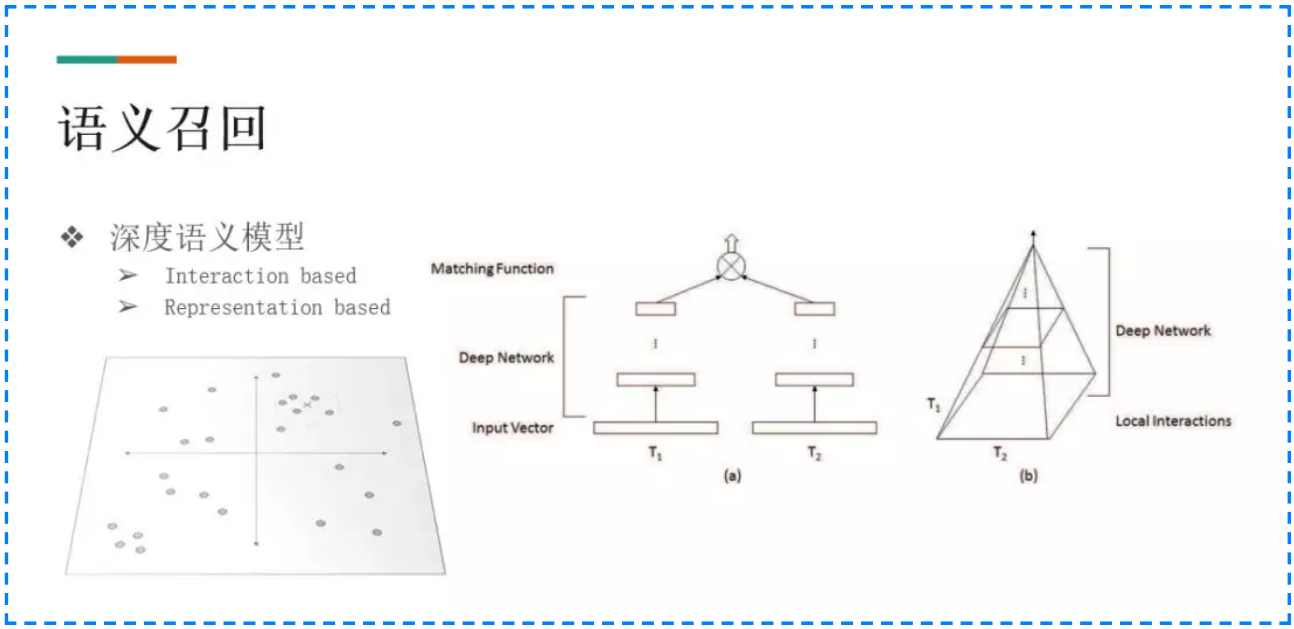
在改写中我们还训练了从 title 到 query 的模型，解决了检索中的重要字段 clicked query 的覆盖率有限和展现偏置两个问题。Clicked query 字段是记录点击过这个文档的 query，就是说尽管这个 doc 不含 query 里的词，但如果有别的用户搜了类似的 query 所点击的 doc，也会将其召回。

记录 clicked query 会遇到点击噪音和展示偏置的问题。因为有点击的文档比没有点击的文档要多一个字段，使得在召回时有一定优势，从而形成正向循环，导致了展示偏置。

通过训练从 title 到 query 的模型，替换掉了 clicked query。这样把每个文档都写有这个字段。虽然模型的效果没有 query 到 query 的模型好，但因为这个模型建在索引里所以容错率会更高。

以上就是基于词的一些算法阶段的过程，最后分享一下语义召回。

语义召回



语义召回就是把 query 和 doc 都表示为向量，然后在向量空间中做最近邻的搜索。相对于图像领域，在 NLP 领域还不够成熟，因为用户输入的表达可能差一个字含义完全不同提高了语义召回难度。

在模型上需要训练深度语义模型，主要分为两种：

❶ 基于 interaction 的深度语义模型

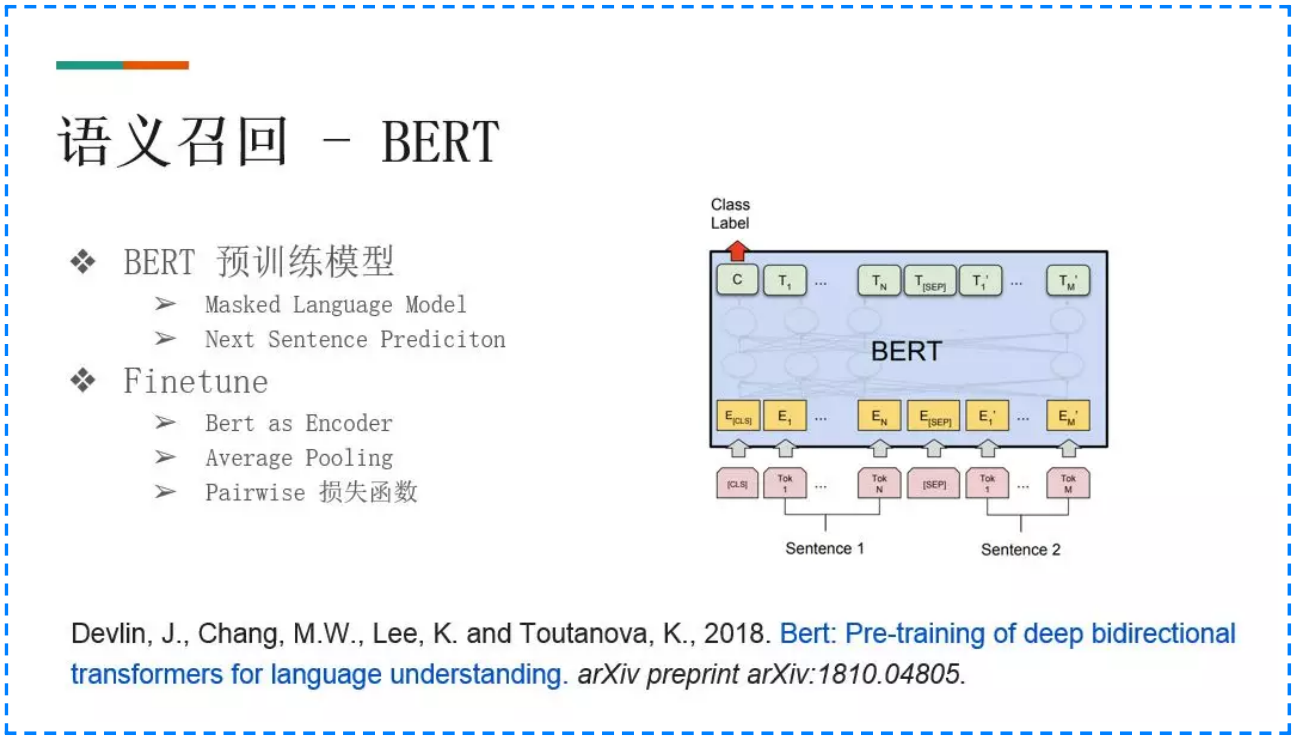
Interaction based 用于判断两个文档的相似性。该方法提前算出 query 里每个词和 titile 里每个词的相似性，得到相似性矩阵。之后直接用例如 CNN 图像卷积的方法处理该矩阵，计算相似度。

❷ 基于 representation 的深度语义模型

Representation 的意思是先分别将 query 和 title 进行网络的处理，分别得到向量，再预测两者相似度的分数。

按照经验来说，interaction based model 效果会好于 representation based model。但是在向量召回这个任务当中，我们需要离线计算好所有文档的 embedding 并建成向量索引，所以语义召回采用的的是 representation based 模型。

语义召回 – BERT 模型



在 NLP 领域，做 embedding 召回难度大，原因为：

❶ Query 长尾，难以捕捉细粒度的差异

❷ 需要泛化

❸ 过多误召回伤害用户体验

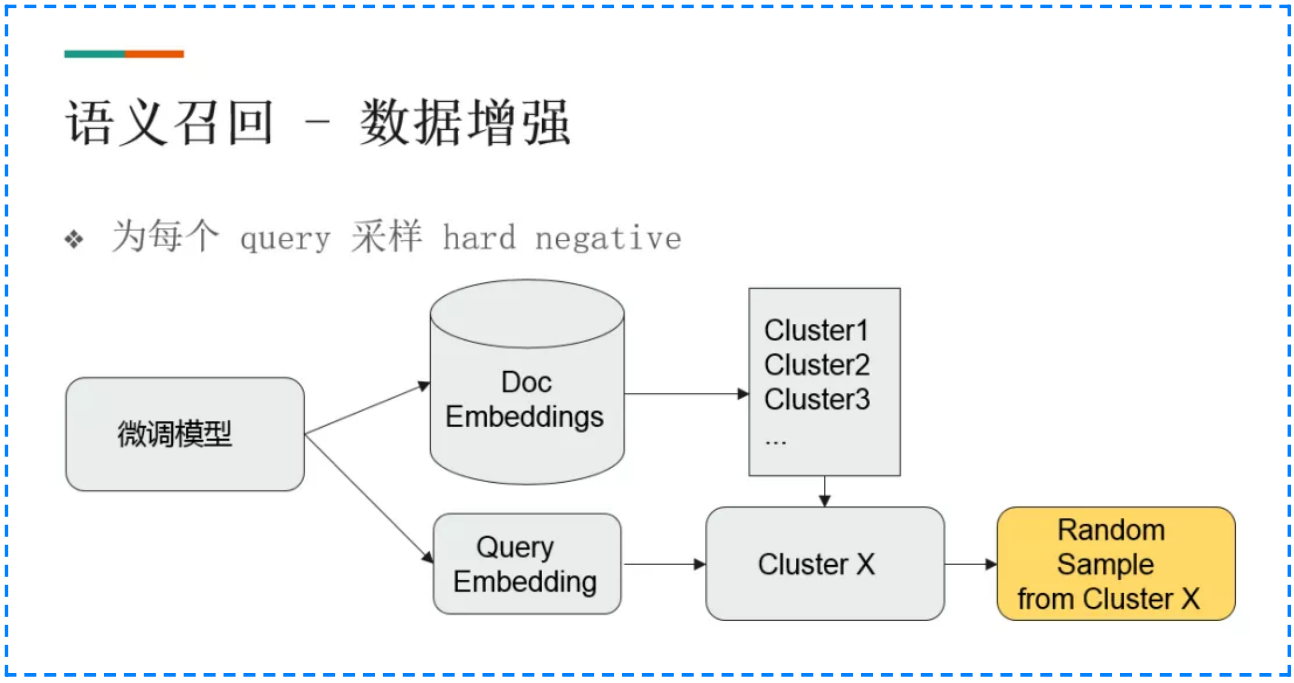
目前BERT 模型带领 NLP 进入了新的时代，借鉴经典的孪生网络结构，我们用 BERT 模型分别做 query 和 doc 的 encoder，接着用 pooling 之后的 query doc cosine 距离当做输出，最后通过 pairwise 损失函数训练模型。在 pooling 方法上，我们也尝试过不同层 pooling，或者在多层上增加 attention 聚合 BERT 多层结果，效果和最后一层所有 token 做 average pooling 相当。损失函数方面，因为我们是做召回阶段的模型，所以参考过雅虎的方法，他们认为在一轮排序阶段使用 pointwise 损失函数的效果会更好，但是在我们的数据上 pairwise 还是会稍好一些。

在 BERT 模型的基础上，我们还做了两项优化：

❶ 优化预训练模型， mask token pretrain

这里主要是参考 百度 ernie 的做法，在知乎的语料上，用中文的分词粒度的数据继续做了 mask token pretrain，使得预训练模型能够更好的适应中文的词关系

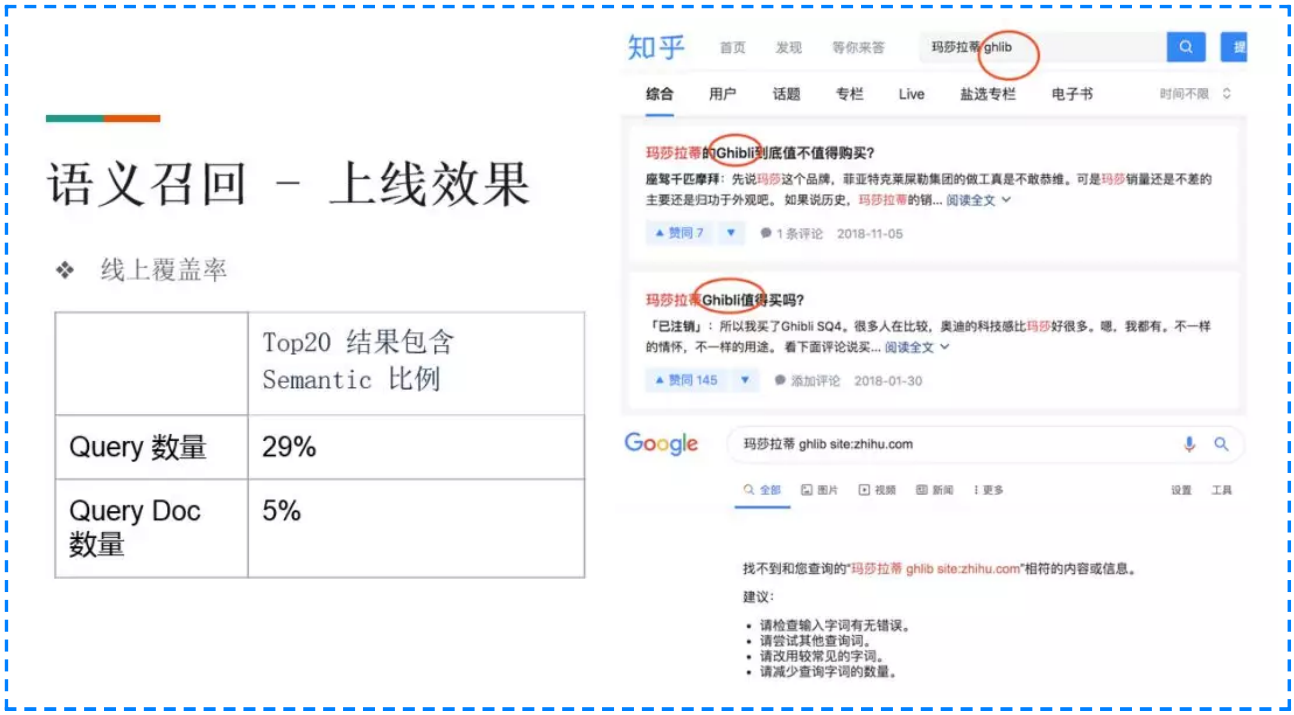
❷ 数据增强，增加一些对于模型来说更难区分的负样本



具体采样方法为：

用原始的脚本数据去微调一个模型，计算 doc 的 embedding 和 query 的 embedding。对 doc embedding 进行聚类，此时 doc 和 query 的 embedding 就在同一空间里，所以

query embedding 就可以对应到某一个类别。这时候即可从 doc 里直接采样一些负样本了。



向量召回上线以后效果提升明显。如图右上方的 case，虽然 query 错误，但仍召回了正确结果。

后续方向

以上是知乎搜索最近一年在召回侧的一些工作，后续的其他方向有：

❶ GAN 在 query 改写的应用

因为之前用的是真实系统计算 reward，代价较高，所以考虑用 gan 来代替真实系统去计算 reward，这样样本数据的计算会速度非常快。

❷ 预训练方面

尝试把预训练模型做小以便用于更多场景中。目前由于参数规模大，时间开销大，导致无法广泛运用。

❸ 进一步探索强化学习在其他场景的应用

如应用于 Query 的 term weight 中。