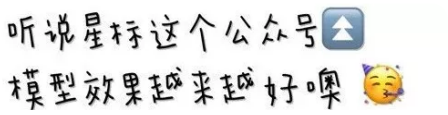
全方位解读 | Facebook的搜索是怎么做的?

原创 一块小蛋糕 NewBeeNLP 2020-10-09

收录于话题
#推荐搜索

4个



> NewBeeNLP原创出品
> 公众号专栏作者@**一块小蛋糕**
> 知乎 | 推荐系统小筑

今天要和大家分享的论文是来自Facebook的『Embedding based Retrieval in Facebook Search』。

不得不说，F家的文章还是一如既往**浓浓的工业风**，这篇论文从工程角度讲解了一个召回的全流程，不管是做语义信息检索召回还是推荐召回都值得认真学习。有过前几个月的embedding召回方向工作后，深觉这篇论文对于做召回的同学来说有非常多可以总结思考的地方。

社交网络中的搜索除了要考虑传统web搜索中的query文本，搜索人所处的上下文也是需要重点考虑的，其中Facebook搜索场景中特有的社交图谱便是典型的一种上下文。尽管向量检索(Embedding_based Retrieval，EBR)已经广泛应用于web搜索，但是Facebook的搜索还是沿用之前的布尔匹配。这篇论文也是从升级Facebook搜索系统开始，总结Facebook将Embedding技术应用到其搜索场景过程中的踩坑经验。

文章中的成果有两个：

- 一个统一的Embedding框架用于构建个性化搜索中的语义Embedding
- 一个基于倒排索引执行Embedding检索的系统

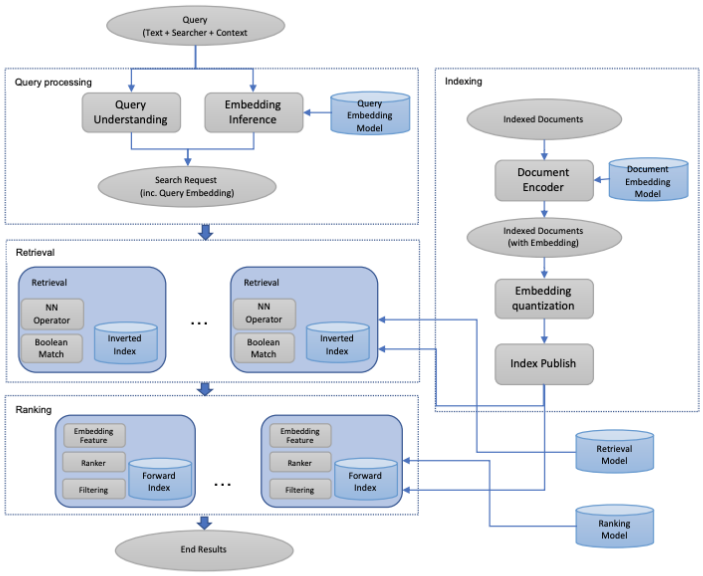


Figure 1: Embedding Based Retrieval System Overview

在进行整个系统的端到端优化过程中，积累了大量的工程优化经验和技巧，如ANN调参，全栈优化等，还有一些如召回模型样本构建，特征工程方面的技巧。

通过引入Embedding技术，使用Embedding表示query和文档，将检索问题转化为Embedding空间的最近邻搜索问题。Facebook搜索中的EBR不仅面临海量数据的问题，如数十亿甚至上百亿的文档，而且需要同时兼容Embedding检索和布尔检索。Facebook搜索还有一个独特的点是用户的搜索意图不仅跟query的文本内容有关，还跟提问者及其所处的环境有关，这一点是比常规的信息检索方向要复杂的多的。

文章从**模型**、**线上服务**、**全栈优化**三个方面介绍了引入Embedding技术过程中遇到的种种问题。

Modeling

作者将搜索中的预取看成召回优化问题，给定搜索的query，及其对应的目标集和模型返回的topK集合，最大化topK结果集中的召回率，其中目标集一般是根据用户点击或人工打分的相关文档。

召回优化问题又可以看成是根据文档和query之间的距离排序的问题，使用cos相似度度量被神经网络编码成稠密向量的query和文档之间的距离，再使用triplet loss作为召回目标学习Embedding编码模型。

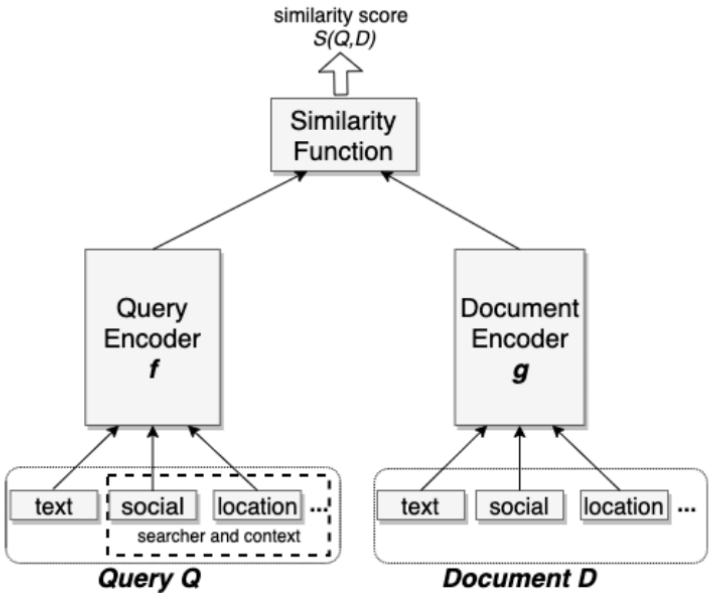
但Facebook的搜索和通常的信息检索不同的地方是除了要考虑文本信息外，还要考虑用户的个性化信息以及用户所处的上下文。因此，文章提出统一Embedding，将查询的query文本和搜索人及其所处上下文融合到一个embedding中。

接下来从统一Embedding模型内容，模型损失函数，训练数据的挖掘和评估指标四个方面介绍Facebook的Embedding模型。

模型内容

首先**「模型内容」**方面，和双塔结构一样，统一Embedding模型由三部分组成：query编码器生成query的Embedding，文档编码器生成文档的Embedding，相似度计算函数生成文档和query之间的打分。Facebook的模型中，query和文档的编码器是共享部分参数的独立模型。使用Cosine相似度作为相似函数，实际的Cosine距离定义为 $1-cos(E_Q, E_D)$ 。

编码器模型的输入包括query和文档的文本内容、社交和其他上下文特征。比如query侧的搜索人位置及其社交连接，文档侧group搜索时用到的小组聚合地址和社交群。原始特征大多为高基类别特征，进入模型的方式都是通过Embedding层，跟普通深度模型一样。



损失函数

其次是**「损失函数」**，基于尽量把正例和负例的距离控制到一个固定距离之上的想法，Embedding模型的损失函数使用triplet loss：

$$L = \sum_{i=1}^{N} max(0, D(q^i, d_+^i) - D(q^i, d_-^i) + m)$$

其中 $D(u, v)$ 表示向量u和v之间的距离度量，m是正负样本之间的最小距离， N是从训练集中挑选的三元组数目。实验中发现margin参数m会导致KNN召回率有5~10%的波动，不同训练任务的最优margin差别也比较大。作者还认为随机采样作为triplet loss的负样本对可以近似召回优化任务，原因是一个正样本匹配n个负样本时，模型会优化n个召回池中选取

top1时的召回率，假设放大到实际服务时的召回池规模为N，近似优化的就是召回topK=N/n的召回率。

训练数据

然后是**「训练数据」**挖掘方面，Facebook基于召回指标验证召回流程中不同正负样本的选择策略。

针对以用户点击为正样本时的负样本选择：

- 从文档池随机选取作为负样本，即easy case；
- 同一次会话中的曝光未点击文档作为负样本，即hard case。

结果表明，曝光未点击作为负样本的召回率远低于随机负样本，约55%的召回率退化。作者认为原因在于全部以hard case做负样本的训练数据和实际召回任务面对的数据分布不一致，实际索引中大多数是和用户query差别很大的easy case。

针对正样本的选择策略：

- 用户点击为正样本
- 曝光即为正样本

实验表明，用户点击和曝光分别作为正样本的召回指标相差不多，添加曝光数据并不能增加额外价值，增大训练数据规模也不能。

评估指标

最后是**「评估指标」**，为了减少线上实验成本，快速评估模型效果并定位问题，文章提出基于索引后的Embedding执行KNN搜索再统计recall@K指标。

特征工程

统一Embedding模型的优势之一就是可以通过融合文本之外的特征提升模型。Facebook的统一Embedding模型相比只基于文本特征的模型，在事件搜索上的召回率可以提升18%，分组搜索的召回率提升16%。统一Embedding模型中用到的特征主要有Text文本特征、位置特征、社交Embedding特征。

- **「文本特征」**：使用字符n元组（character n-gram），相比Word n-gram有两个好处，词表大小会比较小，训练更有效，还有就是缓解query和文档侧的OOV问题。而且在character 三元组的基础上增加Word n-gram表示仍然能稳定的额外提升1.5%的召回

率。由于Word n-gram的基数超高（约352M），需要使用hash减小词表大小，结果表明，尽管会有hash冲突，仍然可以有额外模型提升。Facebook中的实体可以是人、小组、页面或事件。对于这些实体，名字或标题是文本特征的主要来源，在实体搜索中，基于文本的Embedding模型在模糊匹配和纠错方面表现的都比布尔匹配更好。

- **「位置特征」**：在本地广告、小组或事件的搜索场景中，位置匹配是很重要的。query侧增加搜索人的城市，地区，国家和语言。文档侧增加管理员打的小组地域标签。再加上前面的文本特征，模型能顺利学习到query和文档间隐式的位置匹配效果。
- **「社交Embedding特征」**：基于Facebook的社交图谱学习用户和实体的Embedding。

线上服务

说完离线建模，再说说线上serving。Facebook的线上服务采用基于ANN的倒排索引搜索，出于两点考虑：Embedding量化后存储需求更小，便于集成到现有召回系统的倒排索引中。使用Faiss库索引向量，再在现有倒排索引表中做高效NN搜索。

量化调优

这里先介绍一下Embedding量化的背景知识，在向量集合中查找与指定向量距离最短的k个最近邻，暴力搜索的时间在海量规模下是不现实的，Embedding量化就是为了解决向量空间搜索的效率问题，具体来说就是将原有的无限的向量空间映射到一个有限的向量集合（codebook）中，通过映射后的向量近似代表原向量。Embedding量化有三种形式：

1. **「聚类量化」**：常用的是k-means聚类，使用聚类后的类簇中心向量近似表示原始向量，类簇个数即为codebook大小。量化后，每个向量都用类簇中心id表示，向量之间的距离用各自所在类簇的中心向量距离近似表示。对应Faiss中的IndexIVFlat。
2. **「乘积量化」**：PQ量化，由于向量不同部分的分布不同，考虑对向量分段量化。将向量分成m个不同的部分，对每个部分进行向量量化，如平均划分维度。最终的codebook大小为每个部分量化codebook大小的乘积，分块量化中的每个部分量化也可以采取聚类量化实现。乘积量化可以大幅降低空间占用，每个向量可以用m个分块聚类簇中心表示。
3. **「粗糙量化+残差量化」**：层次量化，即Faiss中PQ量化IndexIVFPQ的实现。粗糙量化使用聚类量化，再对残差结果进行细粒度的乘积量化，具体来说就是，每个向量先进行粗糙量化划分到某个粗糙聚类簇里，对应某个类簇标识id，然后计算残差向量（向量-聚类簇中心向量），对残差向量进行分块，执行细粒度分块残差量化，每个残差向量m块，对应m个类簇标识id。因为原始向量聚类后，不同类簇可能分布的比较分散，样本数极度不平衡，而计算残差可以缩小这种不平衡。层次量化对于每个doc生成两个字段：粗糙量化id，残差量化code，用于实时检索使用。其中，向量距离计算过程如下：

首先是每个向量y的量化结果：

$$y = q_c(y) + q_p(y - q_c(y)) = y_C + y_R$$

其中， y_C 是粗糙量化结果， y_R 是残差量化结果。然后是计算查询向量x和y之间的距离：

$$d(x,y) = ||x-y||^2 = ||x - y_C - y_R = ||x - y_C||^2 + ||y_R||^2 + 2y_C y_R - 2xy_R$$

第一项是x和y的粗糙量化结果向量的欧式距离，第二第三项与查询向量x无关，可以提前计算好，第四项是x和y的残差量化结果的内积。

Facebook的Embedding量化使用的是层次量化：粗糙量化加残差的乘积量化。粗糙量化阶段的参数有num_cluster和具体量化算法IMI和IVF。乘积量化阶段是pq_bytes和不同的乘积量化变种PQ，OPQ，带PCA的PQ。还有一个nprobe参数，表示查询query向量可以属于多少类簇，决定了查询近邻时需要计算多少个粗糙类簇向量。文章中也介绍了ANN调参过程中的一些经验技巧：

- 调试召回率的同时关注扫描的文档数。相同num_cluster和nprobe的情况下，对比不同的粗糙量化算法，可以发现类簇是极度不平衡的，导致查询时扫描的文档数各不相同。因此增加扫描的文档数作为调试指标。
- 模型效果有较大提升时需重新调试ANN参数：ANN的表现和模型特征有关联。模型变化较大时需要重新调试
- 优先尝试OPQ算法。OPQ算法的表现总是优于其他算法。但OPQ会对Embedding执行翻转，因此需要重新调试num_cluster和nprobe参数才能得到相似的文档扫描数。
- pq_bytes设置为 $d/4$ 。乘积量化将原本的d维向量压缩成x个字节的编码。x越大精度越高但内存和延时也越高。经验表明， $d/4$ 最佳。
- 在线调试nprobe、num_cluster、pq_bytes。线上部署多组参数，对比验证。

系统实现

Facebook原本的检索引擎支持的是布尔检索，为了避免创建新的系统，扩展了现有引擎的embedding字段以支持nn查询操作（nn <key>:radius <radius>）。索引阶段，每个文档的Embedding被量化成一项（粗糙类簇）和一个payload（量化后的残差向量）。查询阶段，nn查询操作会被改写成一个or运算项，or的内容是和查询Embedding最近的nprobe个粗糙类簇，对于匹配到的文档再使用payload验证半径限制。这里作者针对NN操作对比了半径查询模式和top-K查询模式，发现半径模式更能平衡系统性能和结果质量，作者给出的理由是半径模式下是一个受限NN搜索而topK需要扫描全部索引。

- 混合召回，支持NN操作符后，在现有的检索引擎中就可以随意组合embedding和布尔项的查询。
- 模型服务，查询 Embedding模型和文档Embedding模型同时训练，独立服务。对于查询，是在线实时推断。对于文档则是spark离线批量推断，再建立前向索引，增加额外的量化和PQ生成倒排索引。
- 查询与索引选择，为了提升查询效率和结果质量，避免过度触发、海量空间占用、无用内容堆积等问题，作者在响应过程中使用了一些规则过滤掉EBR会表现差的查询，比如用户搜索之前搜索过或点击过的东西，或者搜索意图完全不同于Embedding模型的训练含义的，只针对月活用户、最近的事件、比较流行的页面和小组做索引选择加快搜索速度。

全链路优化

Facebook搜索排序是个复杂的多阶段排序系统，每层都是对前一层的结果提取精华。检索层是最底层，也是EBR应用的地方，其结果会被后续的排序层排序过滤。每一阶段的模型都是根据前一层的结果数据分布来做优化的。因此，当前的排序系统是根据当前的检索层设计模型，会导致新的检索结果不会得到最优排序。文章提出两个方法解决这个问题：

- Embedding信息作为排序特征之一：既能帮助排序识别来自EBR的新结果，又能提供所有结果的通用相似度。文中对比了使用query和文档Embedding的Cosine相似度，哈达玛积和原始Embedding三种选项，Cosine相似度特征总是最优的。
- 训练数据反馈循环：为了解决EBR精确率低的问题，增加人工打标的流程，使用标记后的数据重新训练模型，提升模型精确率。

为了进一步提升EBR的效果，文中尝试了两个方向：hard样本挖掘和Embedding模型集成。

hard样本挖掘

hard样本的说法来自于CV的分类任务，搜索召回中没有类别的概念，无法直接应用CV的hard样本挖掘方法。FB尝试了两种hard样本挖掘的方法：hard负样本挖掘和hard正样本挖掘。

- **「hard负样本挖掘」**：模型分析时发现Embedding搜索的topK结果，通常具有相同的名字，而且即使提供社交特征，模型也并不总能把目标结果排的更靠前。因此猜测模型并不能很好的利用社交特征，很有可能是因为随机挑选的负样本区分度太明显了，名字大多都是不一样的。因此需要挖掘一些和正样本相似度比较高的负样本。文中尝试了在线和离线两种生成hard负样本的方法：在线是指在训练过程中，把同一个batch内部的其他正样本作为当前样本的负样本，实验证明这一方法能在所有指标上大幅提升模型质量，且每个

正样本生成两个hard负样本是最优的，但这个方法会被batch内的正样本池限制，有可能足够hard的样本数不够；离线的方式则是个迭代的过程，根据规则从每个query生成topK的结果中选择hard负样本生成新的训练数据，重新训练模型。

- **「hard正样本挖掘」**：从搜索行为日志中挖掘搜索失败的会话的潜在正样本作为hard正样本。使用这样的hard正样本能够只以4%的点击数据规模就达到同样的效果。联合hard正样本和曝光数据训练还能进一步提升模型效果。

不过hard样本作为easy样本的补充比单独使用hard样本有更好的效果，因为线上实际召回的数据分布中还是easy样本居多。

Embedding模型集成

首先是通过实验发现，相比easy样本有助于表示检索空间，hard样本能够帮助提升模型精度。随机负样本训练出的模型能模拟实际的检索数据分布，优化的是较大K时的topK召回率，当K比较小时，效果不佳。但如果以精度优化模型，比如以曝光未点击做负样本或离线hard负样本，都会使得模型擅长小数据集内排序而不适合召回任务。因此作者考虑以多阶段的方式融合针对不同难度的样本训练模型，即第一阶段关注模型召回率，第二阶段专注于区分第一阶段中比较相似的结果。文中试验了两种模型集成方式：权重拼接和模型级联，而且这两种都证明是有效的方式。

- **「权重拼接」**：不同模型可以并行训练，针对每个query和文档对，每个模型都会得到一个相似度分值，根据每个模型分配到的权重得到这些分值的加权和作为最终的相似度分值。实际系统中通过将query和文档的embedding各自规范化之后，将模型的权重乘到query侧或文档侧再按照正常累加的方式计算相似度。这里文中的对比实验发现，曝光未点击作负样本的数据训练到的模型与离线挖掘的hard负样本训练到的模型相集成后比单独使用曝光未点击更得到线上比较理想的召回率提升。

- **「模型级联」**：与权重拼接的并行方式不同，模型级联是串行的，在第一阶段的结果上执行第二阶段的模型。同样，曝光未点击做负样本并不能得到理想的召回率提升。而离线hard负样本虽然能明显提升召回率，但依然不如权重拼接的表现。这里文中还尝试了另外一种级联方式：使用文本Embedding预筛选出结果再使用统一Embedding对结果排序，取topK后返回。结果表明这种级联比单独使用统一Embedding召回有明显线上提升。

总的来说，曝光未点击作负样本对排序是关键，而对召回并无太大用处。

embedding召回思考

这篇论文中可以学习到的经验：

- 训练样本构造：召回中的样本选择尤其负样本选择是比其他环节更能决定召回效果的部分，不同难度的正负样本构造都值得尝试。
- 该用规则的地方不要犹豫：不必追求模型的完美化，在模型擅长的地方用模型，模型不擅长的地方用规则。
- embedding使用方式：直接量化搜索或者加权拼接后量化搜索。
- embedding来源选择：融合文本、上下文、社交环境等信息，对于有图片或音视频数据的场景下考虑多模态融合更合适。
- embedding训练方式：并行多模型训练，根据样本难度分阶段训练。

一起交流

重磅推荐！NewBeeNLP目前已经建立了多个不同方向交流群（**机器学习 / 深度学习 / 自然语言处理 / 面试交流 / 大厂内推** 等），赶紧添加下方微信加入一起讨论学习吧！

- END -

往期推荐

关于Scikit-Learn你（也许）不知道的10件事

2020-08-03

ICLR2020 | 深度自适应Transformer