EMBEDDING 在大厂推荐场景中的工程化实践

这篇文章解读YouTube、Airbnb、Alibaba的三篇经典论文，总结Embedding在工业界的一些用法和技巧，这三篇论文亮点众多，提供的经验非常值得我们去细细品味和借鉴。这篇文章几乎把三篇论文的重要内容都进行了解读和总结，需花点时间去研读，文中难免有错误和理解不对的地方，欢迎指正讨论！

作者：卢明冬

原文链接

<https://lumingdong.cn/engineering-practice-of-embedding-in-recommendation-scenario.html>

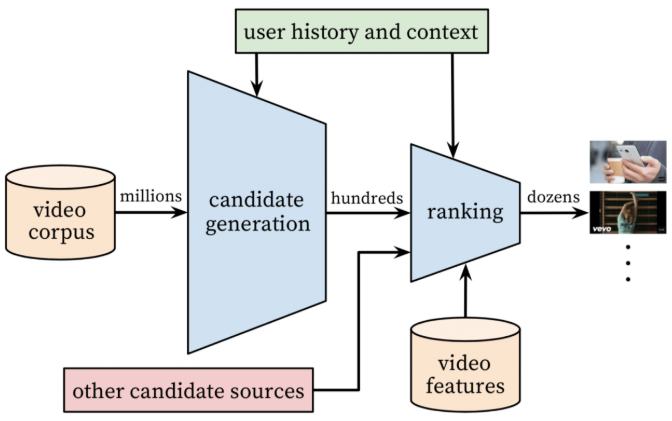
<https://mp.weixin.qq.com/s/8Mx8CznNBlJ6adlwXbcHXQ>

# 1、YouTube

论文题目：Deep Neural Networks for YouTube Recommendations

YouTube 的这篇论文发表在 2016 年 9 月的 RecSys 会议上，是一篇非常经典的推荐系统工程论文。在这篇论文中，不但尝试将DNN用在推荐系统上，更是讲述了众多工程经验，这些经验非常值得我们在工程实践中借鉴学习，相信如今很多互联网公司的推荐系统都曾或多或少地有参考过这篇论文。

YouTube 的推荐系统架构使用经典的召回（Match 或 Candidate Generation）和排序（Ranking）经典的二级架构：

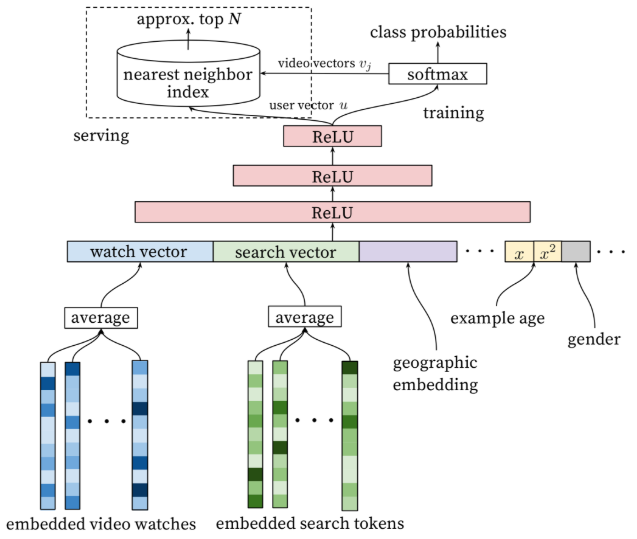


* 召回阶段通过 i2i/u2i/u2u/user profile 等方式完成候选视频的快速筛选1，这个过程将百万（millions）候选集降低到了几百（hundreds）的量级。
* 排序阶段对召回后的视频采用更精细的特征计算user-item之间的排序得分，作为最终输出推荐结果的依据，这个过程将候选集从几百（hundreds）的量级降低至几十（dozens）的量级。

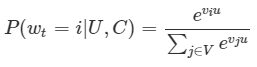
不管是召回阶段还是排序阶段，模型架构上都是 DNN 的基本结构，不同的是输入特征和优化目标不同。

## 1.1召回阶段

召回阶段模型（也称候选生成模型，Candidate Generation Model）的架构如下图：



这个模型虽然是一个深度神经网络，但其实「可将DNN看作是普通的矩阵分解(MF)的一种泛化」，这里把推荐问题建模成一个“超大规模”多分类问题。即在时刻t，为用户U（上下文信息C）在视频库V中精准地预测出下一次观看（next watch）视频i的类别（每个具体的视频视为一个类别，i即为一个类别，观看完成的视频则视为正例），用一个Softmax公式表示这个条件概率：



其中，向量是 <user, context> 信息的高维 Embedding，而向量则表示每个候选视频 j 的 Embedding。所以 DNN 的目标就是在用户信息和上下文信息为输入条件下学习用户的Embedding向量，用公式表达DNN就是在拟合函数：



接下来，我们总结在召回阶段使用的一些工程技巧。

**1、输入融合多种信息，使用全网数据而不是只使用用户浏览数据**

模型输入不仅仅只有用户浏览历史，还加入了用户的搜索历史，人口统计学信息，地理信息以及其余上下文信息，这些输入信息大部分都是先将变长的稀疏 ID 特征转换成固定宽度的 Embedding 向量化表示（固定维度的输入才能输入深度神经网络，也便于参数学习以及模型的工程化），除了 Embedding 类的特征，还有一些简单的二元特征以及连续型特征（比如用户性别、登录状态、年龄等）归一化到 [0, 1] 区间上的实数值，论文很巧妙地将多种信息拼接（concat）成一个特别宽的输入向量，然后传给后面的 DNN 模型的隐藏层，这种方式的带来的好处是任何连续型特征和类别型特征都可以很方便地加进模型中，并且可以维持一致的宽度。

另外，多种信息的融合也带来几个额外的好处：

* 可以缓解冷启动问题；
* 为模型带来更全面的特征信息，提高用户Embedding的表征力度；
* 为推荐场景提供一些探索功能。

**输入特征再展开来讲，里面又涉及很多特征处理经验：**

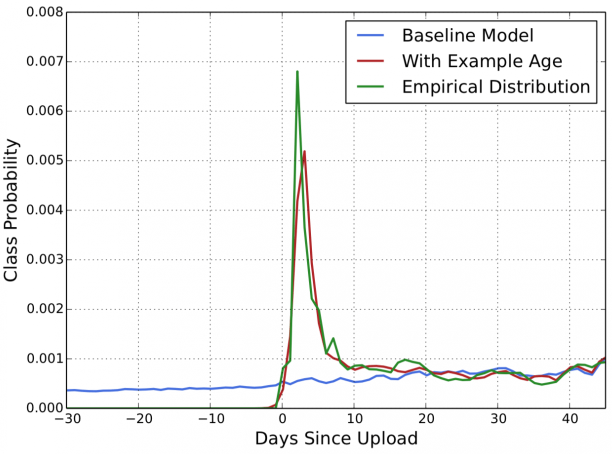
* **观看历史**：这里是通过非端到端的预训练方式先做了 Embedding 之后才作为 DNN 的输入，训练 Embedding 用的是用户历史观看视频的序列，得到视频的 Embedding 表示。而输入 DNN 模型的时候，是将用户观看视频历史所对应的视频 Embedding 序列进行向量平均得到固定维度的 watch vector 作为 DNN 的输入，注意这里向量平均的方式可以是简单的算术平均，也可以是加权平均（可根据视频的重要程度和时间属性等进行加权）。
* **搜索历史**：搜索历史的处理和观看历史的处理方式类似，有些特别的是，这里先把历史搜索的关键词 query 分词得到词条 token，训练得到 token 的 Embedding 向量，然后将用户历史搜索的 token 所对应的 Embedding 向量进行 （加权）平均得到固定维度的 search vector 作为 DNN 的输入，这样便能反应用户搜索历史的整体状态。
* **人口统计学信息**：性别、年龄、地域等，简单的二元特征以及连续型特征会归一化为 [0, 1] 区间上的实数值。
* **其他上下文信息**：设备、登录状态等。
* **样本年龄（Example Age）**：这是一个很特别的特征，它通过捕捉用户兴趣的短期变化，用来拟合用户对新视频的偏好变化带来的偏差。YouTube 算法团队发现，用户特别偏爱那些新上的视频，尽管这些视频的内容可能与他们的偏好并不相关。一些新视频的流量攀升，一方面可能是用户本身喜欢新鲜感带来的一次传播，另一方面热门视频也存在着病毒式的二次传播现象，由此可见视频流行度其实是一个非常不稳定的分布，而机器学习很难捕捉到这种不稳定的变化。

由于机器学习模型通常是基于历史样本训练，来预测将来的行为，因此机器学习模型常常会表现一种出趋向于过去行为的隐含偏差 (implicit bias)。同样的，在推荐系统模型上也存在这样的问题，推荐系统预测出的基于视频库产生的多项式分布，其实反映的是训练时间窗口内也就是近几周可能观看视频的平均情况，而不能真正获取到未来的信息。

为了修正这一偏差，YouTube 算法团队在模型中加入了一个样本年龄（Example Age）的特征，文中并没有精确的定义什么是 example age，不过按照文章的说法，example Age 指的是训练样本的 age，应该是以样本产生的时间（样本记录或样本日志标记的时间点）作为样本的 “出生” 时间，表示的是用户点击某个视频的时间，而不是视频发布的时间，样本产生时间到训练当前时间的距离作为样本年龄（example age）。除了修正趋于过去行为的这一偏差，这个特征还可以一定程度上来捕捉用户的兴趣的短期变化，比如模型可以学到 example age 越小的样本，也就是离训练这一时刻越近的样本，越符合用户的最近的兴趣，从而在模型中权重更大。也因此，在线 serving 的时候这个特征会置零，表示模型在训练时间窗口的最末尾做预测，也就是要预测用户在这一刻的兴趣。

这是一种非常通用的来消除机器学习系统固有偏差的方法。另外训练和在线区别对待也非常值得借鉴，一些单侧特征，训练的时候把bias影响特征加进去，在线的时候则置0，比如美团分享的消除广告position bias（位置给广告点击带来的偏差）也是类似的做法。

论文对这个特征进行了验证，对于给定视频，Example Age 这个特征能够很好的把视频的新鲜程度对流行度的影响引入模型中，对比如下：



从上图中我们也可以看到，在引入 “Example Age” 这个特征后，模型的预测效力更接近经验分布；而不引入 Example Age 的蓝线，模型在所有时间节点上的预测趋近于平均（baseline），这显然是不符合客观实际的。

**2、为每个用户生成固定数量的训练样本**

这也是一个非常实用的训练技巧，为每个用户固定样本数量上限，平等地对待每个用户，避免loss被少数活跃用户代表，能明显提升线上效果，这个方法有点类似协同过滤中对活跃用户的惩罚。

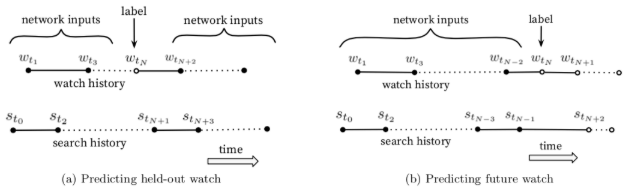
**3、丢弃用户观看视频以及历史搜索query的时序特征**

如果过多考虑时序的影响，用户的推荐结果将过多受最近观看或搜索的视频的影响，导致越推越窄，推荐结果也失去了多样性。所以为了综合考虑之前多次搜索和观看的信息，YouTube 丢掉了时序信息，将用户近期的历史纪录对应的 Embedding 向量做平均或加权平均。这里丢弃时序信息，只是一种经验性的实践，YouTube 团队后来也尝试过一些使用 RNN 处理时序信息的方案。

**4、输入数据只使用标签之前的历史信息，避免产生数据穿越**

用户观看视频的行为存在着明显的不对称共同浏览（asymmetric co-watch）现象，即用户浏览视频的时候，往往是序列式的，有些前后观看行为甚至存在一些因果关联，比如用户通常会在开始看一些比较流行的，后来会逐渐找到细分的视频，而对于那些连续剧集以及关联程度比较大的视频，这种现象表现得更加明显。

因此YouTube的推荐模型的训练样本构建并没有使用类似Word2Vec那样将前后上下文作为输入信息、中心词作为Label的留一法（held-out），而是将未来的信息完全分离，因为我们是预测用户的下一次观看，而非去预测一个随机留一的视频观看(a randomly held-out watch)。二者的区别见下图：



而实际上，传统的协同过滤类算法，都是隐含地采用上面图(a)的heldout方式，其实这种方式泄露了将来的信息(future information)，忽略了用户观看视频存在着的时序性。论文中也提到，上面图(b)的方式在线上A/B test中表现更好。

**5、召回模型在离线训练时，使用负例采样来解决Softmax的低效问题**

这个优化在上一篇文章中也提到过，传统的Softmax在大规模分类问题中效率非常低下，它通常有两种优化替代方式，一种是负例采样（negative sampling），一种是层次 Softmax（hierarchical softmax），这里使用的是负例采样，并且按照重要性加权（importance weighting）来对采样进行校正（calibration）。因此对于每个样本有正负两种Label，最后的学习目标是最小化交叉熵损失。论文中提到，YouTube召回模型会抽样上千个负样本，这种方法可以比传统的Softmax快100倍，他们也尝试了另外一种层次Softmax的优化方式，但效果不如负例采样好。

**6、召回模型在线Serving过程中采用一种最近邻搜索的方法来提高效率**

因为召回阶段并不需要精确的打分校准，为了提高性能，YouTube在线召回阶段并没有直接采用训练好的模型进行预测，而是线上直接利用最近邻搜索的方法，根据User向量来为用户快速查找出最相关的N个视频。近邻搜索（NN）通常分为精确近邻搜索和近似搜索，精确搜索常见的方法是基于树结构的最近邻搜索，如经典的KD树算法，近似搜索也就是近似最近邻搜索（Approximate Nearest Neighbor，ANN），近似搜索搜索精确度降低但搜索时间减少，非常适合一些场景的工程应用，常见的近似近邻搜索方法有哈希散列和矢量量化，经典的局部敏感哈希(Locality-Sensitive Hashing，LSH)算法就是一种哈希散列方法，论文中可能用的就是这种方法，可以用现成的工具faiss来实现。

其实在亿级人脸识别的场景下也是用的类似 LSH的方法来实现快速搜索，另外注意，因为 LSH 是在同一个空间里找近邻，而这里其实是求向量内积，所以这里需要归一化到单位向量，这两个操作就是完全等价的。这也是人脸Loss讨论里提到的特征归一化和权重归一化。

最近邻搜索会涉及User Embedding和Video Embedding两个Embedding，我们分析一下它们是怎么得到的。

* User Embedding：这个很好理解，其实就是把最后一层隐层输出作为用户向量；
* Video Embedding：这个比较难理解，也存在一些争议。从论文以及我查找的资料来看，这里应该是用Softmax的权重来表示Video的Embedding向量的，论文中有提到"The softmax layer outputs a multinomial distribution over the same 1M video classes with a dimension of 256 (which can be thought of as a separate output video embedding)"。

那么这个时候，就出现了两个Video Embedding，第一个是输入时的根据观看历史在 session中的共现视频序列训练出的Embedding，第二个是NN侧输出，也就是最后一层 Softmax的权重，作为Video Embedding的代表，只是比较有疑问的是，在线阶段的这个向量还会更新吗？我个人觉得在线Serving的时候就不更新了，毕竟是Item Embedding，没必要那么频繁地更新，而如此看论文图中的那条连向数据库的线，就是将Video Embedding向量存入数据库以方便线上得到实时用户向量进行近邻搜索。

从这里来看，此处和矩阵分解又非常相似，模型最后可以得到User Embedding和Video Embedding，在训练时，Softmax概率最大的正好对应User Embedding和Video Embedding向量内积值最大的那个，所以预测的时候直接计算内积（可以转换等价于LSH），也就没必要再用Softmax具体计算概率了。

$ User Embedding 和 Video Embedding 是否在同一空间？

@做最闲的咸鱼

和dssm类似，都是通过内积限制两个embedding在相同空间，在CF中可以通过矩阵分解得到user和video的向量表示，这里最后的softmax就是在做广义矩阵分解，模型最后一层隐层就是user embedding，通过u\*v得到video的概率，v就是video embedding，只不过这里用来作为softmax层的权重。

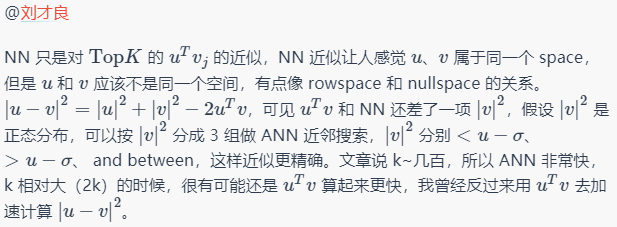
这里说的softmax层是dense+softmax激活函数，假设最后一个hidden layer维度是100 代表user embedding，输出节点维度200w表示videos，全连接权重维度就是[100,200w]，而hidden layer与每一个输出节点的权重维度就是[100，1]，这就是一个video对应embedding，计算一个video的概率时是u\*v，即两个100维向量做内积，是可以在一个空间的。

@张相於

整体来说咸鱼的回答是正确的，但我觉得不完全准确，因为用户向量和视频向量其实不是一个空间，但是是两个相关联的向量空间，所以可以通过内积大小来度量相关性。更好理解的一个例子是word2vec中的两类向量，一类是词的表示向量，也就是进入输入层的向量，另一类向量是预测词的向量，也就是作为权重的向量。换句话说，每个词都有两个向量，那么如果说这两类向量在一个空间中，那就有矛盾了：一个空间中同一个词怎么会有两个不同的向量表示呢？所以我认为这是两个不同的向量空间，但不是无关的两个空间，两个空间通过softmax函数连接在一起。使用时用KNN也是从用户空间中的向量寻找在视频空间中与之最相近的几个向量，并不是同一空间中的KNN。但由于两个空间通过softmax连接起来，所以这样做是OK的。

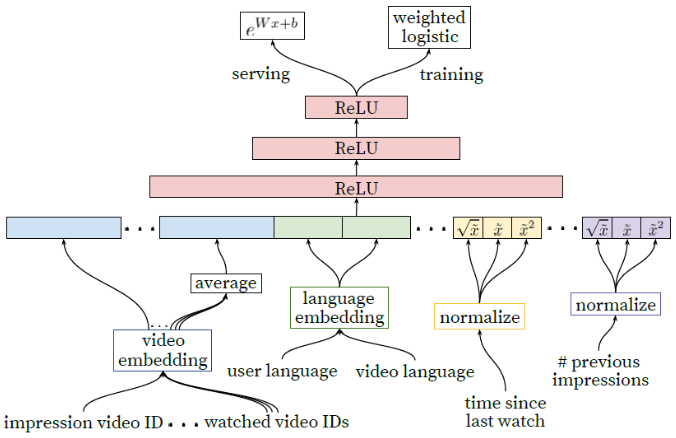
还有一个角度可以佐证两者不是同一空间：如果是同一空间，那么空间中任意两个向量的内积含义应该是一样的。但两个视频空间中的向量V1和V2计算内积，体现的是两个视频的相似性，而一个视频向量V1和用户向量U1计算内积，其含义是用户对视频发生观看行为的概率。这显然是矛盾的。

因此，我倾向于认为视频和用户的向量是两个不同的空间，但通过softmax和训练过程联系在了一起，导致不同空间的内积计算可以表示用户和视频的相关性。



## 1.2排序阶段

排序阶段的模型结构和召回阶段非常类似，可见下图：



* impression video ID：喜欢的视频ID列表
* watched video ID：观看的视频ID列表
* user language：用户语言
* video language：视频语言
* time since last watch：上次观看的时序特征
* #previous impressions：长期兴趣特征

与召回阶段不同的是，排序阶段需要处理计算的数据量仅仅是百数量级的，为了提高预测精度，排序阶段使用了更多精细的特征。除此之外，排序阶段本身就可以整合多源召回，上面提到的召回模型可能仅仅是一种召回策略，通常召回阶段的来源往往很多。

我们来看看排序阶段这篇论文又提到了哪些经典的工程经验。

**1、以观看时长作为优化目标，而非简单地用CTR或者播放率（Play Rate）**

具体而言，论文采用了每次曝光预期播放时间（expected watch time per impression）作为优化目标。确实，观看时长更能反映用户的真实兴趣，而类似CTR的指标则带有迷惑性，尤其是一些靠标题党或者视频封面来吸引人的视频，很容易诱导用户点击，但其内容不一定是用户真正喜欢的。

优化目标的设定其实非常重要，是算法模型的根本性问题，也是最初指明方向的一个过程，如果方向错了，后面再怎么努力也会和真正的目标南辕北辙。目标的确定，除了要考虑业务目标，也要考虑到商业目标，比如 YouTube 推荐的视频被用户看得时间越长，消费用户的注意力越多，就越有商业广告的价值。目标的设定和调整，也可以根据线上A/B test进行微调。

**2、排序阶段重要的特征工程**

深度学习模型虽然能够减少一部分特征工程的工作，但有些原始数据根本就不能直接输入到前馈神经网络（Feedforward Neural Network，FNN），与图像、NLP 不同的是，推荐系统对特征工程格外依赖，好的特征能够起到非常关键的作用，就比如召回阶段提到的 Example Age。因此，在推荐场景下我们仍然需要花费很大一部分精力来做特征工程，将原始数据转换成有用的特征。其中最主要的挑战是如何建模用户时序行为（temporal sequence of user actions），并且将这些行为和要排序的item相关联。

* YouTube发现最重要的特征是描述用户与商品本身或相似商品之间交互的特征，这与 Facebook 在2014年提出LR + GBDT模型的论文（Practical Lessons from Predicting Clicks on Ads at Facebook）中得到的结论是一致的。

因此YouTube考虑了用户与视频频道（或主题）的关系，包括数量特征，即浏览该频道视频的数量，以及时间特征，即最近一次浏览该频道距离现在的时间。这两个连续特征的最大好处是具备非常强的泛化能力。另外除了这两个偏正向的特征，用户对于视频所在频道的一些PV但不点击的行为，即负反馈Signal同样非常重要。

* 把召回阶段的信息，比如推荐来源和所在来源的分数，传播到排序阶段同样能取得很好的提升效果。
* 为了产生很好的响应式推荐，通常会在推荐的时候引入“搅动（churn）”的功能（连续的请求不会返回相同的列表），而使用描述视频历史曝光频率的特征，可以对“搅动”起到很好的辅助效果，比如一个用户最近被推荐了某个视频，但没有观看它，接着模型将自然地在下一页加载时降级该曝光（impression）。
* 对类别特征进行Embedding向量化

NN更适合处理连续特征，因此稀疏的特别是高基数空间的离散特征（类别型特征可能的取值非常多，这种特征也称为高基数类别特征）需要Embedding到稠密的向量中。每个唯一的ID空间（比如user、item、vocabulary）都具有一个单独学到的Embedding空间，一般来说Embedding的空间维度基本与唯一值（去重后的值）数目的对数接近。这些Embedding 通常会在DNN模型训练前构建，并形成简单的look-up table。

YouTube在对Video进行Embedding的时候，直接将大量长尾Video进行了断掉。在基于点击曝光的频率排序后，只选择了TopN的视频进行Embedding，其余全部置为0向量。其实这也是工程和算法的一个权衡（trade-off），截断掉大量长尾可以节省online serving中宝贵的内存资源，当然低频video的Embedding的准确性不佳是另一个“截断掉也不那么可惜”的理由。这个经验也是值得借鉴的，除了置为0向量，也可以替换为所属类别的 Embedding，或者用一个固定的Embedding统一表示，类似NLP中用UNK(unknown)表示长尾词。

另外，和召回阶段一样，在输入时，多值类别特征的Embedding先进行平均处理。

重要的是，相同ID空间的类别型特征，也共享着底层的Embedding。例如，存着单个关于视频ID的全局Embedding，供许多不同的特征使用（曝光的视频 ID，该用户观看的最近视频 ID，作为推荐系统“种子”的视频ID等等）。尽管共享Embedding，但依然需要每个特征独自输入到网络中，以使得上面的层可以学到每个特征的特定表征(representation)。共享嵌入（sharing embeddings）对于提升泛化、加速训练、及减小内存等相当重要。

* 对连续特征进行归一化

众所周知，NN对输入特征的尺度和分布都是非常敏感的，实际上，基本上除了Tree Based的模型（比如 GBDT/RF），机器学习的大多算法都如此。归一化方法对收敛很关键，论文推荐了一种排序分位归一到 ［0, 1）区间的方法，即，累计分位点，该积分与特征值的分位数的线性插值相近似。

除此之外，我们还把归一化后的的平方根和平方作为网络输入，以期使网络能够更容易得到特征的次线性（sub-linear）和（super-linear）超线性函数。通过引入特征的不同阶来引入了特征的非线性。

**3、排序模型没有采用经典的Logistic Regression当做输出层，而是采用了Weighted Logistic Regression。**

这样做的原因主要是为了融合优化目标，因为模型使用期望观看时长（expected watch time per impression）作为优化目标，如果简单使用LR就无法引入正样本的观看时长信息。因此采用weighted LR。

YouTube使用几率（odds）的方式来引入权重信息，相当于对模型sigmoid层作了一个小小的改动。

关于几率，我们在《Logistic 回归》一文中其实已经有过详细介绍，而且Logistic回归本身就是从几率推导出来的，所以Logistic回归又称对数几率回归。我们简单回顾一下。

一个事件的几率，是指该事件发生的概率与该事件不发生的概率的比值，用公式表示：



取对数得到 logit 函数：



logit函数是广义线性的连接函数，因此令：



稍作转换，就得到了我们熟悉的sigmoid的函数：



这里将观看时长Ti作为正样本的权重，负样本权重指定为单位权重1，这样将权重引入几率后，由于正样本对预估的影响提升了 Ti 倍（权重是 Ti），而负样本没有变化（权重是 1），因此对于指定样本，其引入权重后的几率是原来几率的Ti倍：



@张轩

@Shaohua Yang

严格的说，Weighted LR中的单个样本的weight，并不是让这个样本发生的概率变成了 weight倍，而是让这个样本，对预估的影响(也就是loss)提升了weight倍。这种影响的反馈，通过推导可以看到，无论是在更新梯度时作用(乘以weight)，还是直接将此样本在训练集里面扩充到weight个，大体上都是一样的效果(更新策略决定)。非weight的odds可以直接看成N+/N-，因为weighted的lr中，N+变成了weight倍，N-没变，还是1倍，所以直接可得后来的odds是之前odds的weight倍（严格的说应该是sum(weighti)倍）。

由于在视频推荐场景中，对于整个视频资源库来说，用户打开一个视频的概率p往往是一个很小的值，因此上式可以简化为：



由于p就是用户打开视频的概率，Ti是观看时长，因此Ti∗p就是用户观看某视频的期望时长，由此便能理解训练阶段的输出层为什么使用Weight LR了。

而对于线上Serving的输出层，则是使用，它又是怎么来的呢？

其实还是几率（odds），我们把logit函数稍微做个转换变可得到：



因此在线上Serving中使用做预测可以直接得到期望观看时长（expected watch time）的近似。

$ 既然预估的是期望观看时长，那为什么不用多分类或者是回归的模型来替代Weight LR？

@张相於

回归有一个问题在于值域是负无穷到正无穷，在视频推荐这样一个大量观看时间为0 的数据场景，为了优化MSE，很可能会把观看时间预测为负值，而在其他数据场景下又可能预测为超大正值。逻辑回归在这方面的优势在于值域在0到1，对于数据兼容性比较好，尤其对于推荐这种小概率事件（rare event）的场景，相比回归会更加适合。而且odds和LR 的公式是同源的，且值域也是非负的，更符合watch time的物理意义。

另一种方案是把观看时间离散化成k个bucket然后做多分类，但是多分类输出粒度不够细，不适合用来做排序。此外多分类的参数数量也比二分类多很多，同样的样本量下训练效果可能不如二分类效果好。

训练Weighted LR一般来说有两种办法：

1. 将正样本按照weight做重复sampling，然后输入模型进行训练；
2. 在训练的梯度下降过程中，通过改变梯度的weight来得到Weighted LR。

$ 这两种训练方法得到的结果有没有不同？

@张相於

这两种训练方法得到的结果是不一样的，比如要抽样10倍，对于第一种方法，就是把一条样本重复10倍，这样优化的过程中，每遇到一条这个样本，就会用梯度更新一下参数，然后用更新后的参数再去计算下一条样本上的梯度，如果逐步计算并更新梯度10次；但对于第二种方法，则是一次性更新了单条梯度乘以10这么多的梯度，是一种一次到位的做法。

直观一些来讲，第一种方法更像是给予一条样本10倍的关注，愿意花更多时间和精力来对待这条样本，是一种更细致的方法，第二种则比较粗暴了，不愿意花太多功夫，直接给你10倍权重。

@王喆

非常准确的理解，但我觉得这两种方法虽然形式上差别很大，但其实结果上差别不太大，因为重复抽样十次相当于算十次梯度再加起来，跟算一次梯度乘以10的结果一样。当然，重复采样逐次算梯度时还要考虑当前最优点移动的影响，但我觉得对于SGD来说影响不会特别大。

@张相於

当前参数点的移动确实会对结果造成影响，尤其是当把这10条样本打散分布在所有样本中时，也就是说不是连续使用这10条样本做优化时，影响应该会更大。但具体对最终效果的影响还是得试验后才知道。

@王喆

确实打散后的影响不好评估，结果确实会不同，还是跑一组AB test来的实在了。

# 2、Airbnb

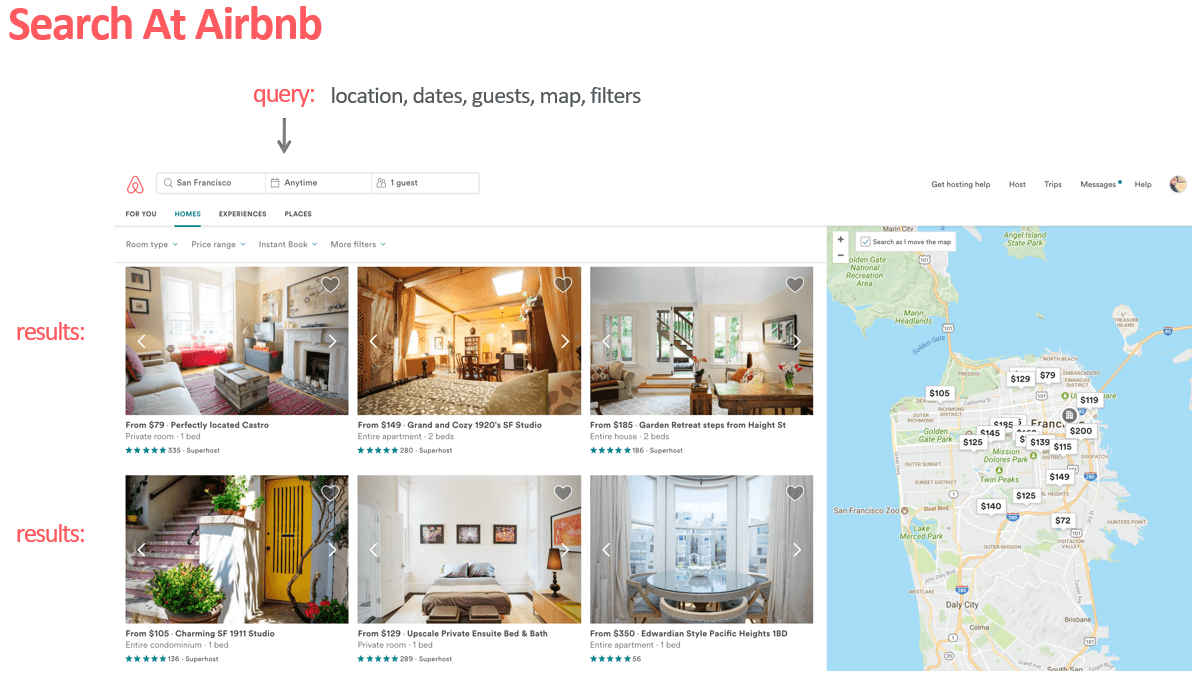
论文题目：Real-time Personalization using Embeddings for Search Ranking at Airbnb

Airbnb这篇论文获得了KDD 2018的Best Paper，和上一小节介绍的YouTube论文类似，也是一篇充满工程实践经验的论文，其中Embedding在Airbnb业务中的整合和灵活应用，非常值得我们解读借鉴。

先来了解一下Airbnb的业务背景。

Airbnb是全球最大的短租平台，平台包含数百万种不同的房源，为了能够帮助用户找到自己心仪的短租屋，Airbnb主要通过两种方式来为用户推荐合适的住房。

一个是搜索推荐，即用户输入地点、日期、人数进行搜索，系统通过复杂的机器学习模型使用上百种信号对搜索结果中的房源进行排序，然后根据你的需求推荐你可能会感兴趣的房源：



另外一个是相似房源推荐，接入在房源详情页底部，会推荐和当前房源相似的房源，以图文滑动窗格的形式展示：



Airbnb表示，平台99%的房源预订来自于搜索推荐和相似房源推荐4，由此可见算法在业务中起到的关键作用。

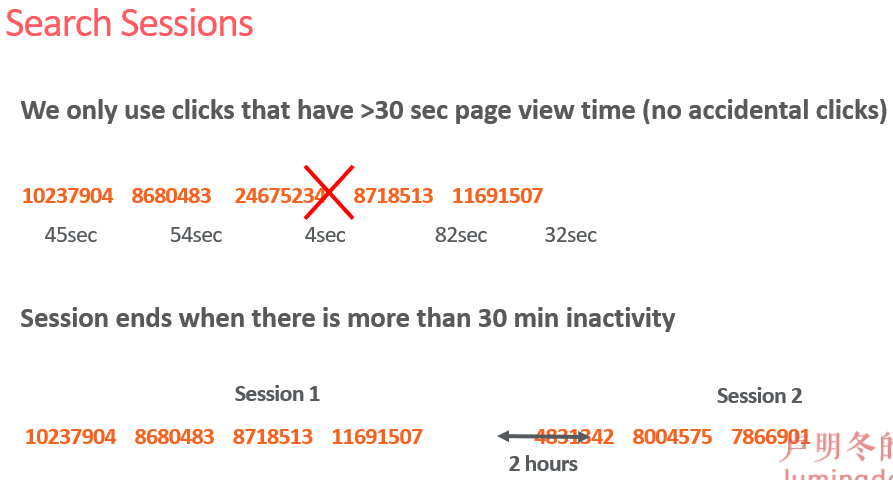
Airbnb论文中将短租房源的item称为listing，其Embedding称为房源嵌入（Listing Embedding），为了与论文保持一致，本文会延用该称谓，大家当作item理解即可。

接下来我们详细看看论文中提到的一些优秀的工程经验。

**1、将用户行为进行合理的Session切分，然后进行Embedding训练。**

这个方法在上一节Graph Embedding也提到过，即基于session的用户行为（session-based）的Embedding。基于session用户行为学习到的Embedding，更容易捕获到用户的短期兴趣。session需要从用户行为历史中获取，因为用户历史行为并不是一次性产生的，用户每次的访问，其偏好都可能不同，因此，需要对用户行为历史进行session切分，这样可以避免非相关序列的产生，也能够清洗噪声点和负反馈信号。

Airbnb采用了click session数据对listing进行Embedding，其中click session指的是一个用户在一次搜索过程中，点击的listing的序列，这个序列需要满足两个条件，一个是只有停留时间超过30s的listing page才被算作序列中的一个数据点（过滤噪声），二是如果用户超过30分钟没有动作，那么这个序列会断掉，不再是一个序列（避免非相关序列）。

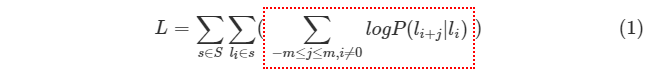


**2、使用负例采样的Skip-gram模型，并根据自己的业务特点进行了两项优化。**

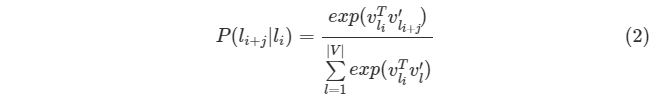
这一部分是论文的一个核心，我们先从目标函数开始，逐步详细推导。

假设，给定从N个用户中获取的S个点击sessions的一个集合S，其中每个session 被定义成：一个关于该用户点击的M个listing ids连续序列，这里的 session就是上面我们提到的已经切分后的session。对于给定该数据集，目标是为每个唯一的listing l\_{i}学习一个d维Embedding表示： ，以使相似的listing在该Embedding空间中更接近。论文中的Embedding维度d设为32，并且在论文末尾进行了试验，验证这样的设置可以平衡离线性能和在线搜索服务器内存中存储向量所需的空间，能够更好地进行实时相似度的计算。

学习Embedding使用的是Skip-gram模型，通过最大化搜索sessions的集合S的目标函数L来学习listing表示，L 定义如下：



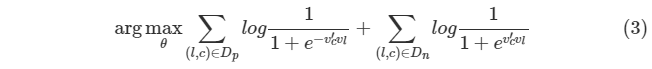
从被点击的listing l\_{i}的上下文邻居上观察一个listing l\_{i+j}的概率，使用 Softmax 定义：



其中和是关于listing l的输入和输出的向量表示，超参数m被定义成对于一个点击 listing的forward looking和backward looking上下文长度，V被定义成在数据集中唯一 listings 的词汇表。

可以发现，上面的方法基本和Word2Vec或Item2Vec的思想一致，当然梯度更新也会受到Softmax性能的影响，因此采用负例采样进行优化。

我们会生成一个positive pairs (l,c)的集合Dp，其中l表示点击的listings，c表示它的上下文，然后从整个词典V中随机抽取n个listings来组成negative pairs (l,c) 的集合Dn。优化的目标函数变为：

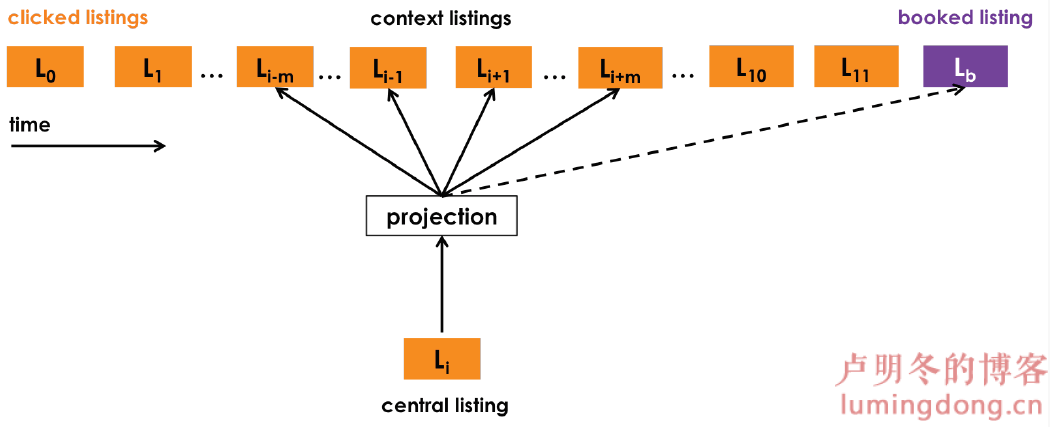


其中要学的参数θ是：和，。优化通过随机梯度上升法（SGA）完成。

公式（1）、（2）、（3）都是最基本的用法，接下来我们来看看Airbnb如何根据自己的业务特点来优化Embedding学习的目标函数。

**优化一：使用最终预订的房源作为全局上下文 (Global Context)**

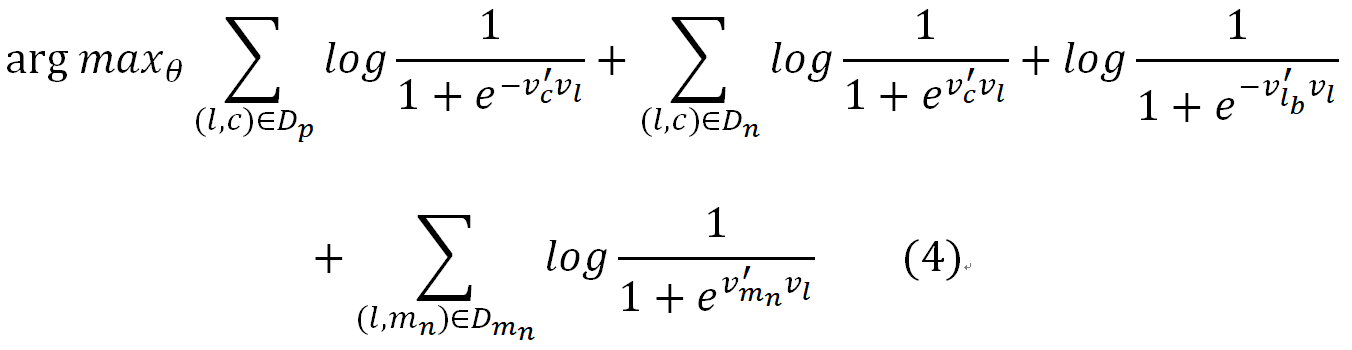
这一优化主要使用以用户预订房源（booked listing）作为结束的会话，加入预定不仅可以预测相邻的点击房源（clicked listings），还会预测最终预订的房源。 当窗口滑动时，一些房源会进入和离开窗口，而预订的房源始终作为全局上下文（图中虚线）保留在其中，并用于更新中心房源（central listing）向量。



**优化二：适配聚集搜索的情况（优化负样本的选择）**

在线旅行预订网站的用户通常仅在他们的旅行目的地内进行搜索。因此，对于给定的中心房源，正相关的房源主要包括来自相同目的地的房源，而负相关房源主要包括来自不同目的地的房源，因为它们是从整个房源列表中随机抽样的，这种不平衡会导致在一个目的地内相似性不是最优的。为了解决这个问题，我们添加了一组从中央房源的目的地中抽样选择的随机负例样本集。论文中将像目的地这种限定因素称之为市场market，其实应用时并不限定于目的地的地理位置信息，还可以拓展到其他限定因素，只要能够从这些限定因素中找到更合适的负例样本添加到目标函数即可。

因此，经过上面的两个优化，最终优化后的目标函数变为：



其中：

* 要学习的参数θ有：v\_{l}和v\_{c}，l,c∈V。
* D\_{p}是一对正例样本对(l,c)，表示（中心房源，相关房源）元组，其向量在优化中会被互相推近
* D\_{n}是一对负例样本对(l,c)，表示（中心房源，随机房源）元组，其向量在优化中会被互相推离
* l\_{b}是最终被预订的房源，被视为全局上下文并被推向中心房源向量
* D\_{m\_{n}}是一对目的地维度的负例样本对(l,m\_{n})，代表（中央房源，来自同一目的地的随机列表）元组，其向量被推离彼此

另外需要注意的一点，上面目标函数中，关于“优化一”的部分并没有加和符号∑，这是为什么呢？

其实公式3和公式4都是公式1的红框部分，即对给定session的优化目标函数，并不是整体目标函数L。而对于每个预定session，只会有一个预定，它不像公式其他几个部分是样本对集合，自然就没有加和符号了。

利用上面所描述的优化目标函数，Airbnb使用超过8亿次的搜索点击会话，训练了450 万个有效房源Embedding嵌入表示，从而获得了高质量的房源推荐和展示。

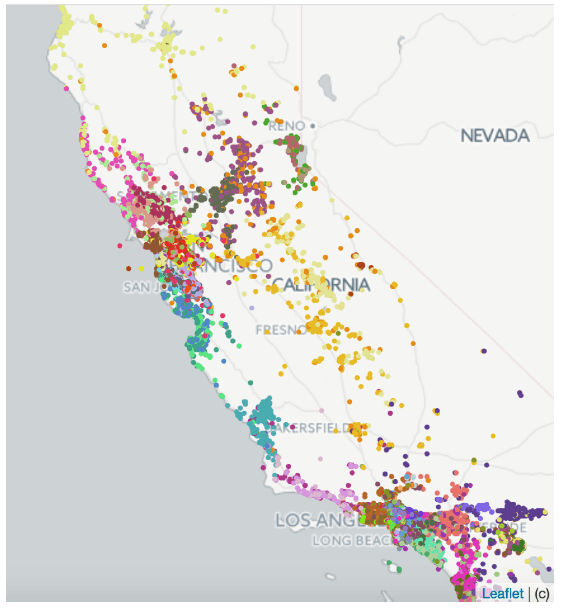
**3、使用相近房源 Embedding 平均值来解决冷启动问题。**

每天在Airbnb上都有新的房源提供。这些房源在新建时还不在训练数据集中，所以没有嵌入信息。为给新房源创建Embedding，Airbnb的方法是找到3个地理位置最接近、房源类别和价格区间相同的已存在的房源，并计算这些房源嵌入的向量平均值来作为新房源的嵌入值。

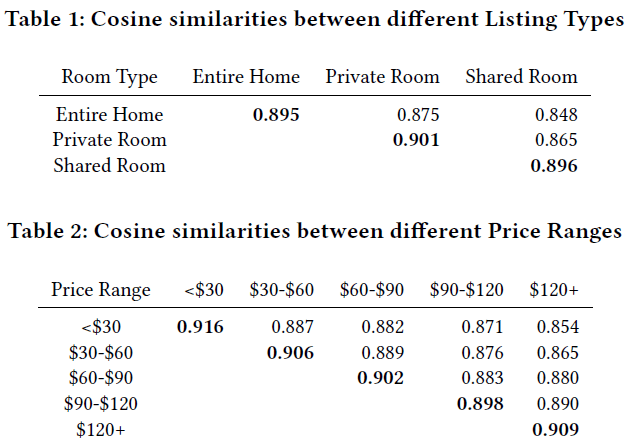
**4、Embedding 的评估经验**

对于Embedding质量的评估，Airbnb使用了很多方法，主要的思想还是通过Embedding 相似度的方法来进行的，论文中提到了以下几种方式：

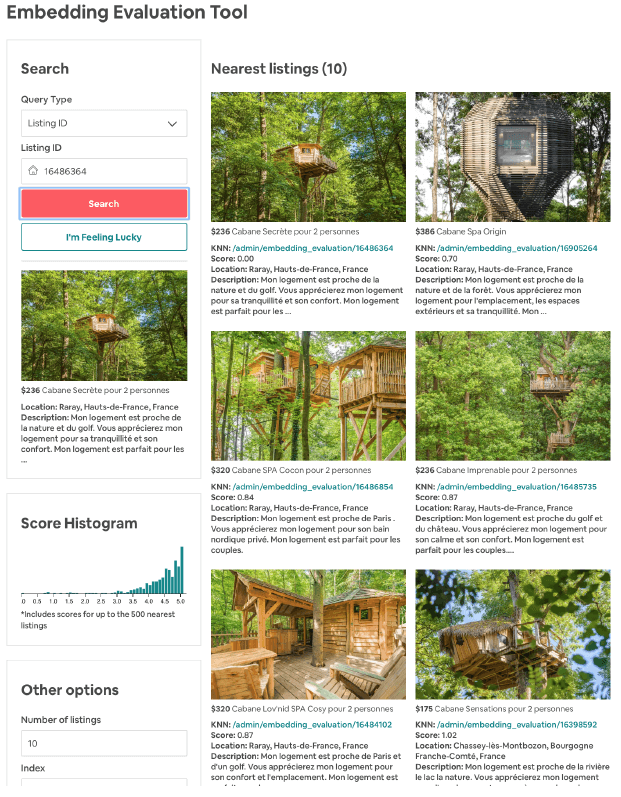
* 评估地理位置相似性是否被包含，对Embedding进行了k均值聚类(k-means clustering)。下面的图显示了美国加州产生的100个聚类，确定了来自近似位置的房源基本能够聚集在一起。



* 评估房源特征相似性是否被包含。论文评估了不同类型（整套房源，独立房间，共享房间）和价格范围的房源之间的平均余弦相似性(cosine similarity)，并确认相同类型和价格范围的房源之间的余弦相似性远高于不同类型和不同价格的房源之间的相似性。因此我们可以得出结论，房源特征也被很好的包括在训练好的Embedding中了。



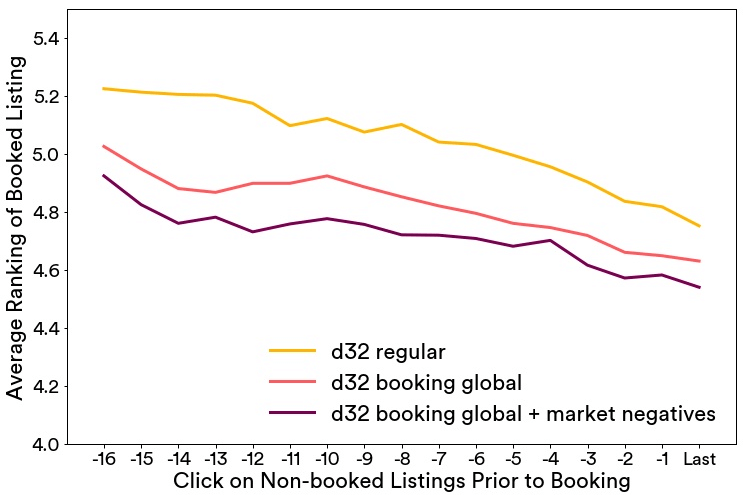
* 评估一些隐式房源特征。部分类型的房源特征（例如建筑风格，样式和感觉）很难提取为房源特征的形式。为了评估这些特性并能够在Embedding空间中进行快速简便的探索，Airbnb内部开发了一个相似性探索工具(Similarity Exploration Tool)，下图的示例表明，通过Embedding能够找到相同独特建筑风格的相似房源，包括船屋、树屋、城堡等，说明即使我们不利用图片信息，也能从用户的click session中挖掘出相似风格的房源。



* 论文还提出一种离线评估方法，测试通过用户最近的点击来推荐的房源，有多大可能最终会产生预订。

更具体地，假设已经获得了最近点击的房源和需要排序的房源候选列表，其中包括用户最终预订的房源；通过计算点击房源和候选房源在Embedding空间的余弦相似度，对候选房源进行排序，并观察最终被预订的房源在排序中的位置。

下图显示了一个此类评估的结果，搜索中的房源根据Embedding空间的相似性进行了重新排序，并且最终被预订房源的排序是按照每次预定前的点击的平均值来计算，追溯到预定前的17次点击。同时，下图也对比了几种Embedding学习的优化方式，可以看到优化起到的效果。



**5、利用User-Type和Listing-Type Embedding来解决数据稀疏问题**

上面学习到的Listing Embedding是使用clicked sessions进行训练的，它们更适合短期 (short-term)、session内（in-session）的个性化的需求，虽然这个Embedding能够很好地给用户展示与在搜索session期间点击的listing相似的listings，但并不包含用户的长期兴趣信息。基于用户长期兴趣的信息对于个性化搜索来说非常有用，比如用户当前session搜索的是洛杉矶的短租屋，但是用户之前在纽约、伦敦住过的短租屋也代表着用户的兴趣偏好。那如何学习含有长期兴趣信息的 Embedding 呢？

Airbnb首先想到的是使用了booking session序列，每个用户按预定(booking)的时间顺序排列的一个listings序列构成booking session，然而想要从booking session 学习到相应的Embedding，会遇到诸多挑战：

1）booking sessions数据集比click sessions数据集要小很多，因为预定本身是低频事件；

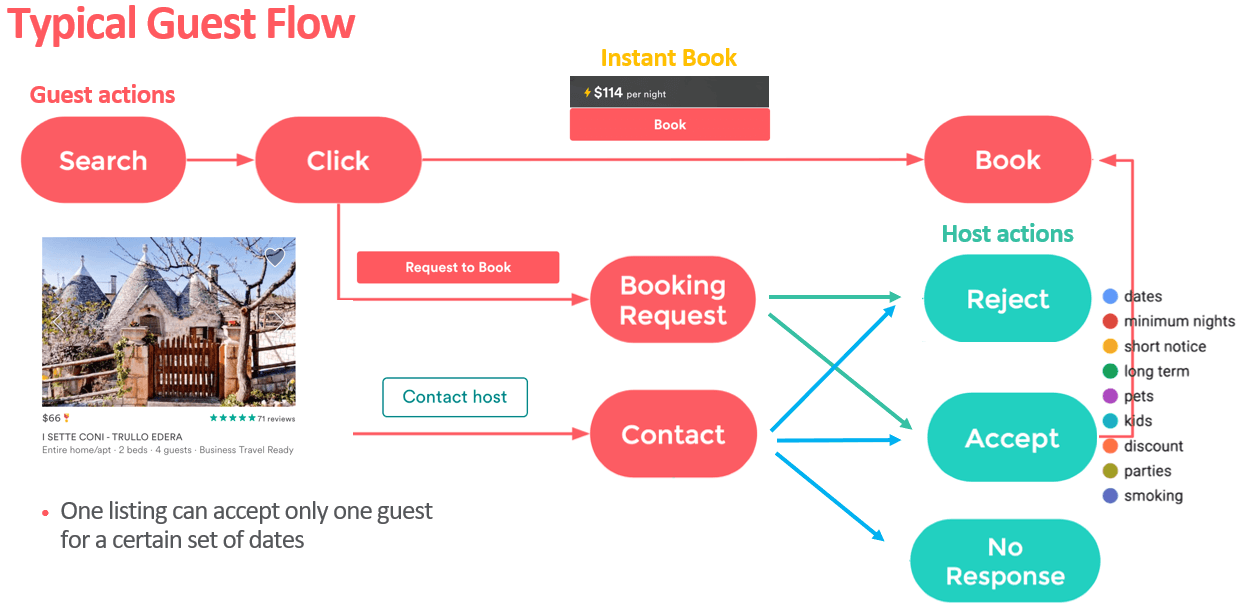
2）许多用户在过去只预定单个listing，我们不能从长度只有1的session中学习Embedding；

3）为了上下文信息中的任意实体学习一个有意义的Embedding，至少需要该实体出现5-10 次，然而在平台中的许多listing\_ids会低于5-10次；

4）最后，由同用户的两个连续预定可能会有很长时间的间隔，这时候，用户偏好（比如：价格点）可能会随很多因素发生了变化，比如职业发展影响等；

5）必须满足供需双方，给租户的最优化推荐还需要考虑房东的对租客的偏好

Airbnb其实是一个双边市场平台，它的房源大都来自于个人的民宿，所以平台也会考虑房东对租客的偏好，和预定酒店等类似业务不同，Airbnb房源的房东是可以拒绝租客的，大致的业务流程如下图所示：



从上图可以看出，booking过程其实存在房东的显式反馈信息（接受或拒绝租客的预定），这个信息也体现出房东的偏好，对房东来说，拒绝预定的部分原因可能是：租客较差的星级评分习惯、用户资料不完整或空白、没有资料头像等等。所以，为了提高租客的预定成功率，平台不仅要考虑租客的租住偏好，还要考虑房东对租客的一些基本要求和偏好，尽量避免给用户推荐的房源被房东拒绝。

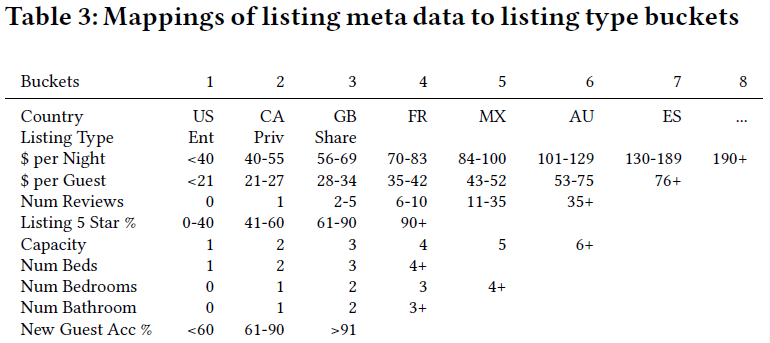
面对这些挑战，如何才能学习到有意义的Embedding呢？

首先，针对第四个挑战，也就是为了捕获用户随时间变化的偏好信息，Airbnb除了Listing Embedding，又加了一个User Embedding的学习。而对于数据稀疏的问题，Airbnb使用了一种非常巧妙的方法，即**基于某些属性规则做相似user和相似listing的聚合，这种方法是通过Type级别来学习Embedding而非ID级别。**

我们接下来看具体的实现过程。

**第一步，通过基于规则（rule-based）的方式定义的属性类型分桶（buckets），将ID类映射到type类。**

* Listing Type 映射



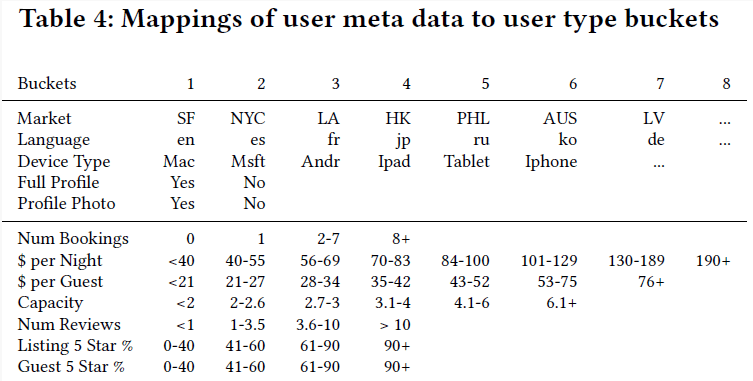
所有listing\_id都可以根据自己的属性通过上面的分桶映射成一个listing\_type，分桶设计是以数据驱动的，要尽可能地覆盖所有的类型。

例如，一个来自US的Entire Home listing（lt1），它是一个二人间（c2），1床（b1），一个卧室（bd2），1个浴室（bt2），每晚平均价格为60.8美元（pn3），每晚每个客人的平均价格为29.3美元（pg3），5个评价（r3），所有均5星好评（5s4），100%的新客接受率（nu3）。因此该listing根据上表规则可以映射为：

listing\_type = US\_lt1\_pn3\_pg3\_r3\_5s4\_c2\_b1\_bd2\_bt2\_nu3

**从listing\_id到listing\_type的映射是一个多对一的映射，这意味着许多listings可能会被映射到相同的listing\_type，因此这把这个过程称之为聚合。**

* User Type 映射



**User-Type Embedding是与Listing-Type Embedding在相同的向量空间学习的，学习 User-Type Embedding主要为了解释用户随时间变化的偏好。**User属性的映射方式同Listing 类似，不过数据除了利用User之前预定记录留下的一些行为信息，同时引入了User本身的一些属性信息，利用前5行的映射一定程度上可以缓解新用户的冷启动问题。

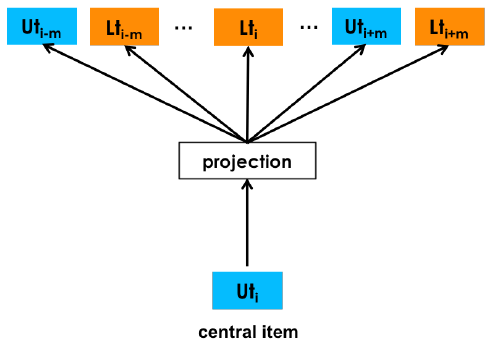
例如，对于一个用户，他来自San Francisco（SF），带有MacBook笔记本（dt1），说英文（lg1），具有用户照片资料（pp1），83.4%平均5星率（l5s3），他在过去有3个预订（nb1），其中关于订单（booked listings）的平均消费统计为：每晚平均价格（Price Per Night）为52.52 美元（ppn2），每晚单客户平均价格（Price Per Night Per Guest）为31.85美元（ppg3），预定房源的平均房间数（Capacity）是2.33（c2），平均浏览数（Reviews，此处存疑，看数量感觉这里的Reviews和Listing中的Reviews可能不一样）是8.24（nr3）、5星好评率（Listing 5 star rating）是76.1%（l5s3）。因此该用户根据上表规则可以映射为：

user\_type = SF\_lg1\_dt1\_fp1\_pp1\_nb1\_ppn2\_ppg3\_c2\_nr3\_l5s3\_g5s3

**第二步，用 type 类数据生成 booking session 序列。**

如何生成同 type 类的 booking session 序列呢？

一种直观方法是直接把user\_type当作原来的user\_id，生成一个由listing\_type组成的 booking session。这种方法虽然能够解决数据稀疏性的问题，却无法直接得到User-Type Embedding。为了学习在相同向量空间中的user\_type和listing\_type的Embeddings，Airbnb的做法是将user\_type插入到booking sessions中，形成一个（user\_type, listing\_type）组成的元组序列，这样就可以让user\_type和listing\_type的在session中的相对位置保持一致了。如下图所示，user\_type和listing\_type相对位置保持一致（不同颜色区分），根据中心Item的类型来选择对应上下文的类型：



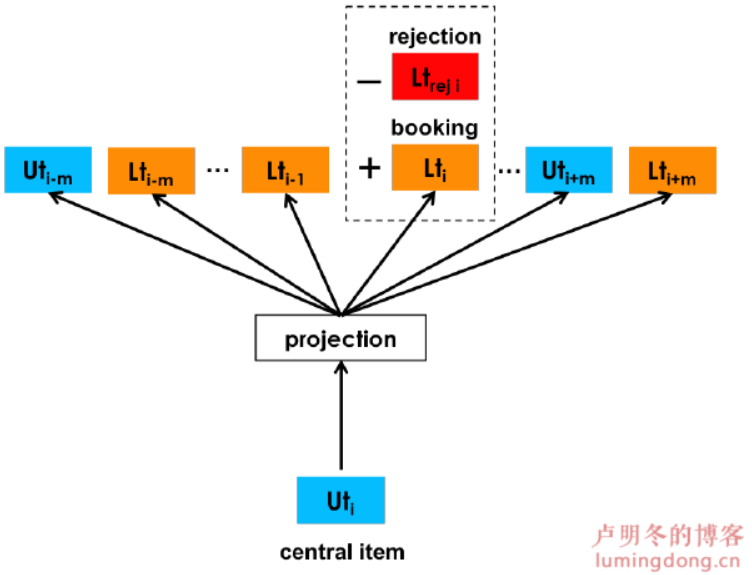
具体来讲，对于给定user\_id按时间排序的booking session，用（user\_type, listing\_type）组成的元组替换掉原来的session中的listing\_id，因此这个booking session 序列就变成了：



这里指的就是预定的 listing对应的listing\_type，指的是该user在预定 listing时的user\_type，**由于某一user的user\_type会随着时间变化，所以每个session虽然由相同user\_id的bookings组成，但对于单个user\_id来说，他们的user\_type在序列中的不同位置则不一定相同。**

**第三步，设计优化目标函数，进行 Embedding 学习。**

有了type类的booking session序列，训练Embedding的方式基本和上面的一致，也是采用负例采样的方式。当然，此处Airbnb也考虑了根据自身业务对负例采样进行进一步的优化工作。由于定义中的booking sessions几乎包含了来自不同markets的listings，因此这里没有必要从相同market中抽样额外的负样本作为booked listing。但为了解决上面提到的第五个挑战，也就是尽量避免推荐被拒，在此处的目标函数中引入了房东拒绝订单（rejection）的显式负例样本。如下图所示，对于同一用户，通过一个正号标记预定成功的listing，通过负号标记被房东拒单的listing。



当在对于另一个listing的成功预定(通过一个正号标记)之后主人拒绝（通过一个负号-标记）。

其实在 User Type 映射表中，就包含了租客的一些个人信息以及行为信息，通过引入房东的拒单信息，就可以捕捉到房东（所对应的房源）对租客的偏好，这样学习到的 Embedding 在进行相似推荐时便可以减小拒绝率，最大化预订成功的概率。

我们对 rejections 看成是显式负样本，以如下方式公式化。除了集合 Dbooking 和 Dneg，我们会生成一个集合 Drej，它由涉及到 rejection 事件的 user\_type 和 listing\_type 的 pairs(ut,lt) 组成。如图 5b 所示，我们特别关注，对于同一用户，当在对于另一个 listing 的成功预定 (通过一个正号标记) 之后主人拒绝（通过一个负号-标记）。

于是，最后改进的新目标函数为：

User Type：