Improving Deep Learning For Airbnb Search

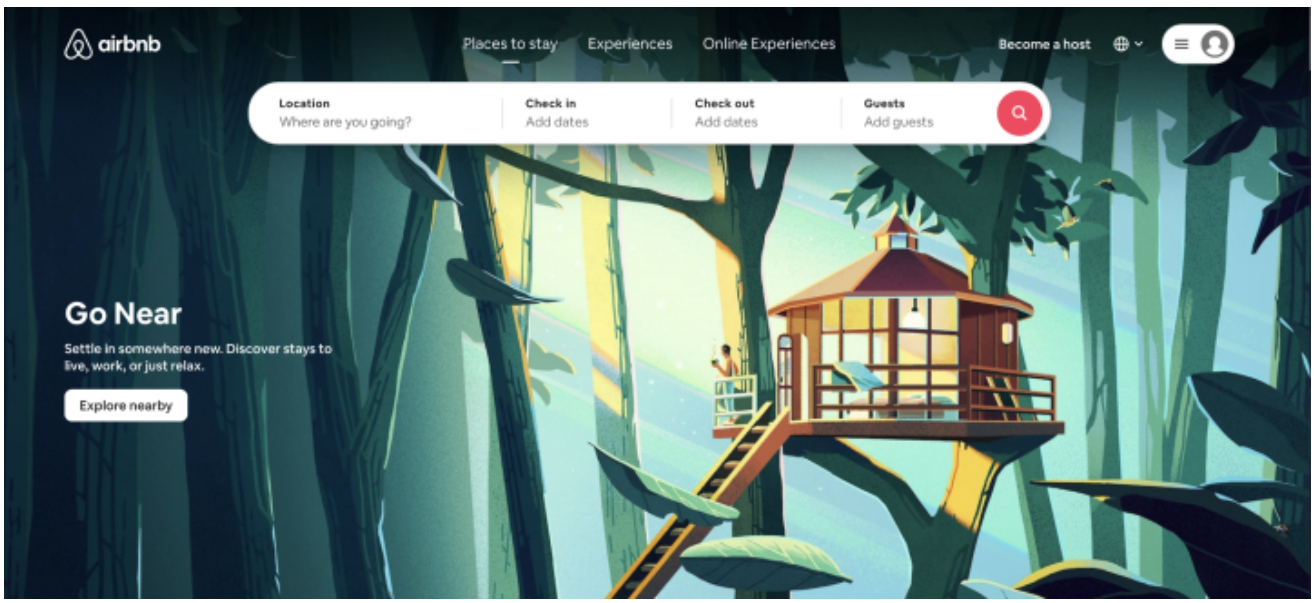
Airbnb将深度学习应用于搜索排序。但是，在你使用了深度学习模型之后，接下来应该做什么？文章提出了三个方面提高模型的性能(ABC)。

- A: architecture, 文章提出了一种新的神经网络排名架构，关注于提升现有的两层DNN
- B: bias,
- C: cold start

1. Introduction

在airbnb的场景下，搜索排序问题可定义为：

- query: 用户查询，包括location, number of guests, checkin/checkout dates
- item: 候选酒店



Airbnb界面

作者提到，他们完全可以把最新、最复杂的模型一个一个实现一遍，然后就完事了😄。

> We could simply pick the best ideas from literature surveys, launch them one after another, and live happily ever after😄.

但是，这样的策略总是带来失望。因为在其他数据集上表现好的方法在我们的应用中可能表现平平。这让Airbnb团队认识到，应该修改他们做模型迭代的方法。除了深入研究核心机器学习技术本身之外，还将重点介绍导致提升的过程和推理。

2. Optimizing the Architecture

在回顾深度学习一系列的进展之后，架构上最直接的优化想法是----加层。但是当我们试图加更多的层时却发现, 模型的测试结果并没有提升。为了解释增加更多的层数还是无法取得有效的提升时，我们想从残差网络，batch normalization中获取想法，但是在离线的测试中NDCG仍然无法提升。我们从中得出的结论是，增加层数是**卷积神经网络**的一种有效技术，但不一定适用于所有DNN。 对于像我们这样的全连接DNN，两个隐藏层就足够了，模

型容量不是我们的问题。

> For fully connected networks like ours, two hidden layers were sufficient and model capacity was not our problem. 💮♀

既然加层行不通，那是不是需要更专业(specialized)的结构？因此，airbnb尝试了引入query和item之间特征交互的模型，例如 wide&deep，其中将query和item的交叉特征放入到wide侧。然后再采用了各种花式的attention模型，attention的目的时是使从query上提取的隐藏层特征注意力集中在item上提取的某些hidden layer上(参考DIN)。但是这种方式模型并没有得到提升，原因是因为一个成功的网络结构往往与其应用的场景(application context)相关。

由于普遍缺乏深度学习的可解释性，因此很难准确地推断出新架构要解决的问题以及解决方法。因此我们只能去猜，现有的架构存在哪些缺陷，然后一通乱改。为了提升成功的可能性，团队抛弃了“下载新论文->复现结构->A/Btest"的模式，采用了一种新的准则(user lead,model follow)，根据分析出模型，实践出真知。

（碎碎念）：确实, 现在大部分推荐/搜索的顶会论文提出的方法，在私有的数据上表现都一般。其实模型架构应该**和数据息息相关**。

2.1 User lead, model follows

首先，我们需要量化用户问题，然后对模型进行调整以响应用户的问题。

Airbnb团队观察到之前模型的成功与**搜索结果的平均价格下降**有关。这预示着迭代越来越接近顾客的价格偏好，而且这个价格比之前模型预估的要低。

> Along those lines, we started with the observation that the series of successful ranking model launches described in [6] were not only associated with an increase in bookings, but also a reduction in the average listing price of search results. this indicated the model iterations were moving closer to the price preference of guests, which was lower than what the previous models had estimated.

上面这段话可能有些难以理解，可以看下面这个图：

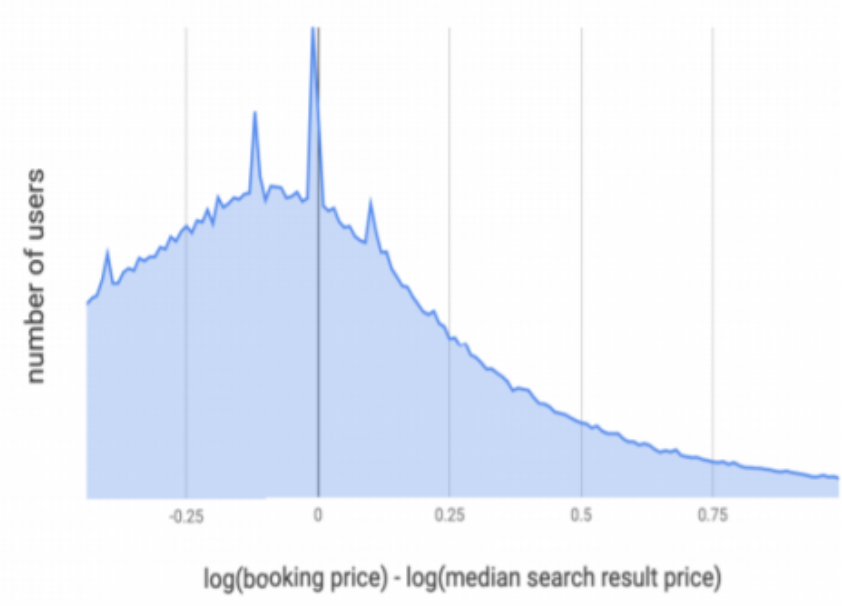


Fig. 1. X-axis shows how the price of the booked listing offsets from the median price of search results for a guest. Y-axis is the number of users corresponding to a price offset.

x轴：log(用户最终选的酒店价格) - log(搜索结果的中位数价格)；y轴：频率

如果用户对一个特征完全没有偏好，那么预订价格会是围绕搜索结果的价格中位数呈正态分布的。但是，上图是明显向左倾斜的(应该叫右偏？)，表明了顾客更喜欢便宜的酒店。那么，我们的ranking model是否真的了解这种 Cheaper Is Better 原则？ 这个是不能确定。

2.2 Enforcing Cheaper is Better

模型缺乏可解释性的原因是我们采用了DNN的模型，它不像LR/GBDT可以计算特征权重。为了让价格这个特征更具有解释性，文章做了如下改变：

- 把price相关的特征移除。修改后的DNN模型记为 $DNN_\theta(u, q, l_{no\_price})$，其中 θ 是DNN的参数，u是user feature，q是query feature，l_{no_price} 是去掉price的listing feature。
- 模型最终的输出为:

$$DNN_\theta(u, q, l_{no\_price}) - tanh(w * P + b)$$

$$P = log(\frac{1 + price}{1 + price_{median}})$$

其中，w 和 b 都是需要学习的参数；$tanh(w * P + b)$ 引入了**单调性**，因为当w>0时，模型的输出能够保证Cheaper Is Better的原则。如下图所示：

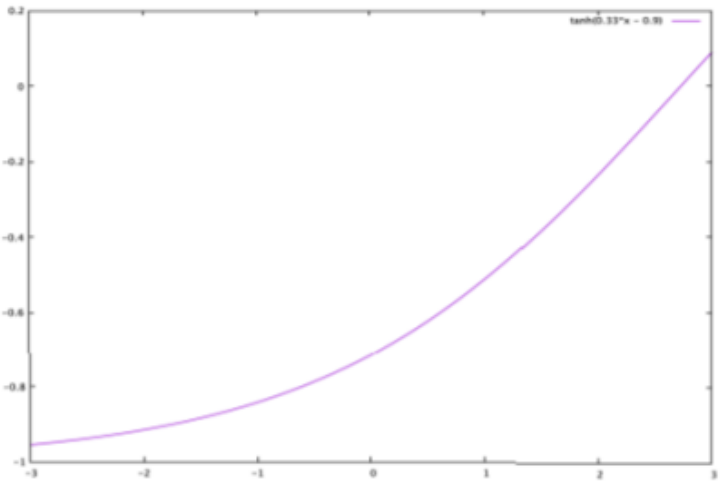


Fig. 2. X-axis is normalized price feature. Y-axis is the value of the $tanh$ term in Equation 1.

x轴：price，y轴：tanh(w*P+b),是单调递增

w, b 两个参数可以提供解释性。最总学到的 $w = 0.33, b = -0.9$.

但是采用上续方式的线上A/B test的表现并不如意。搜索结果的平均价格下降了5.7%。但是下单率下降了1.5%。这可能是因为price特征和其他特征有比较强的**特征交互**，将price特征isolate出来会导致模型的**欠拟合**。在训练集和测试集上NDCG都下降了，这印证了模型是发生了欠拟合。

2.3 Generalized Monotonicity

为了保留cheaper is better，同时让价格特征和其它特征进行**交互**，文章就开始研究一些对输入特征保持单调的架构(DNN architectures that were monotonic with respect to some of its inputs)。所以文章设计了一个这样的解决方案，将price特征抽取出来，单独和price无关的特征交互。

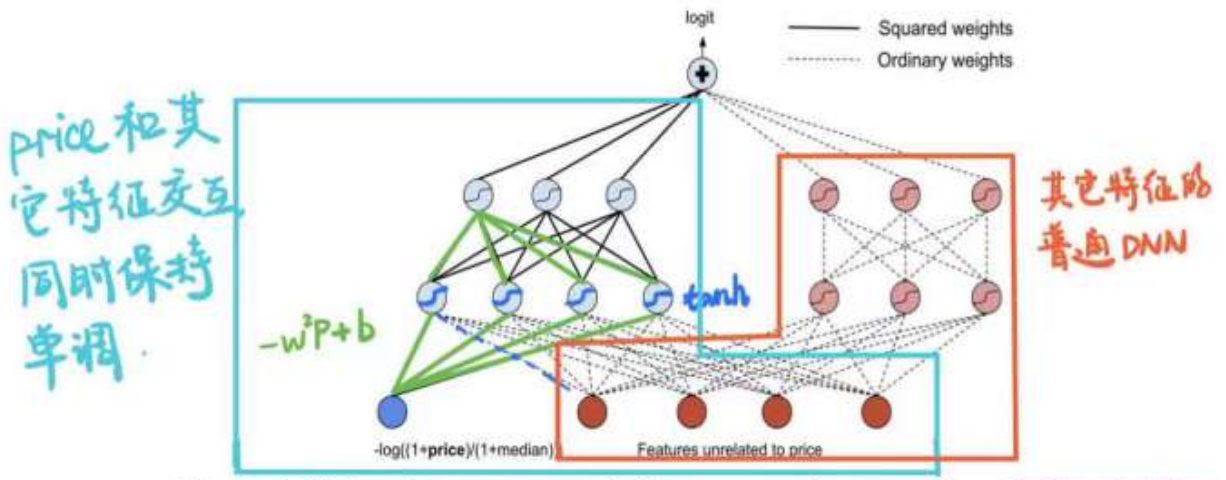


Fig. 3. DNN architecture partially monotonic w.r.t price. Bold solid lines indicate weights that are squared, dashed lines indicate ordinary weights.

实线：权重平方；虚线：正常权重

- 左侧输入的 $-P = -log((1 + price)/(1 + median))$是随着price而单调递减的
- 对于隐层采用权重的平方的形式： $w^2 * (-P) + b$，能够保证单调性。这样，第一隐层的输入就是随价格递减的。
- 使用tanh激活函数能够保证单调性
- 总之，左侧塔的输出是随价格而单调递减的，同时还引入了price和其他特征的交叉。

但是这样的解决方案在线下测试时，预订率有1.6%的下降。这是因**为过于严格的价格下降**导致的。

2.4 Soft Monotonicity

既然过于强制的价格下降会导致模型失准，那么能不能有一个**没那么强制**的方案呢？答案是有的。

我们知道，在搜索的时候经常会用**pairwise的损失**，即训练样本包含同一个query下的**<一个正例(booked)，一个负例(not booked)>**，我们希望给正例比负例更高的分数。

```
def get_loss_op( positive_logits , negative_logits ):
  """ Create the loss op to be minimized.
  """
   logit_diffs = positive_logits - negative_logits
  xentropy = tf.nn. sigmoid_cross_entropy_with_logits (
       labels =tf. ones_like ( logit_diffs ),
       logits = logit_diffs )
  loss = tf.reduce_mean(xentropy)
  return loss

# Booked listings as positives , not booked as negatives
loss = get_loss_op (booked_logits , not_booked_logits )
```

Table 1. TensorFlow™ code for pairwise booking loss.

👆这个损失函数的意思：我们希望logit_diffs（给正例的打分logits-给负例的打分logits）越接近全1向量越好，交叉熵函数就是拿二者做的比较。

为了增加价格的影响，论文引入了第二标签，就是在训练样本中标注哪个item是便宜的、哪个是贵的。这样, loss就会变成两个损失的和，其中alpha 作为超参数调节相关性和价格之间的权重。

```
# Booked listings as positives , not booked as negatives
booking_loss = get_loss_op (booked_logits , not_booked_logits )
# Lower priced listings as positives , higher priced listing as negatives
price_loss = get_loss_op ( lower_price_logits , higher_price_logits )
# Total loss a linear combination with a hyperparameter
loss = alpha* booking_loss + (1 - alpha)* price_loss
```

Table 2. TensorFlow™ code with price loss added.

这样做的意义是，我们希望lower_price_logits-higher_price_logits越接近1越好，即给lower-price的item打分比high-price的item打分要更好才好，这样虽然没有直接让打分随price单调递减，但是也给了一个很强的暗示。

在线上的A/B test中，发现价格减少了3.3%，但是下单率也减少了0.67%。

2.5 Putting Some ICE

上文中提到了一个自相矛盾的问题: 价格下降了但是用户却不喜欢了。为了增强搜索的可解释性，文章使用了individual conditional expectation的想法，**一次只关注一个query的搜索结果**，比较不同的price对模型打分的影响（保持其他feature不变，这个需要依赖一个假设，即price和其他feature不相关）：

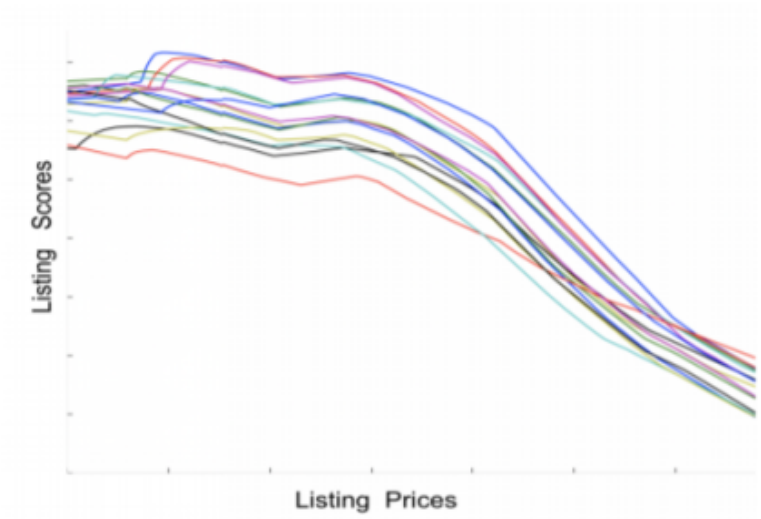


Fig. 4. ICE plot of listing scores (y-axis) vs listing prices (x-axis). Each curve represents a listing in the search result.

x轴：price; y轴：得分，每个颜色就是一个query的结果

从图中可以看出，模型已经学到了cheaper was better。这说明压低价格的结构导致了失败。

2.6 Two Tower Architecture

一个新的想法是，模型已经深刻的理解了 cheaper is better, 但是它没有理解 the right price for the trip. 为了深刻理解这一点，模型需要更加关注一些query feature(比如location), 而不是关注item feature.毕竟，不同location的人们对price这个特征的偏好很不一样，对于一些大城市（LA, San Diego）,人们甚至愿意去预订更贵的酒店：

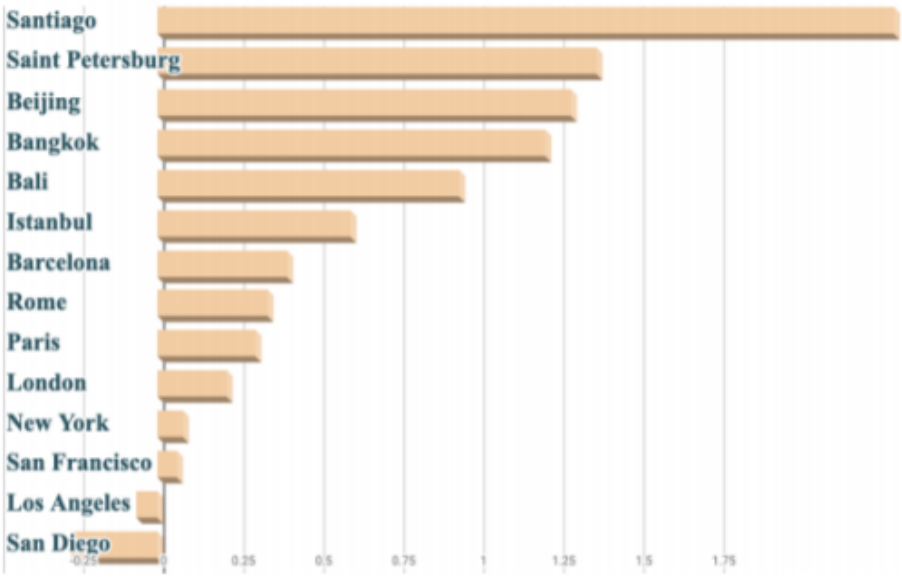


Fig. 5. Average difference between the median price of search results for a guest and the price of the booked listing, split by cities

x轴：搜索结果的价格中位数 - 预订的价格

这样就延申出了下一代的模型--三塔模型（其实这个不就是召回的常规做法吗！！）

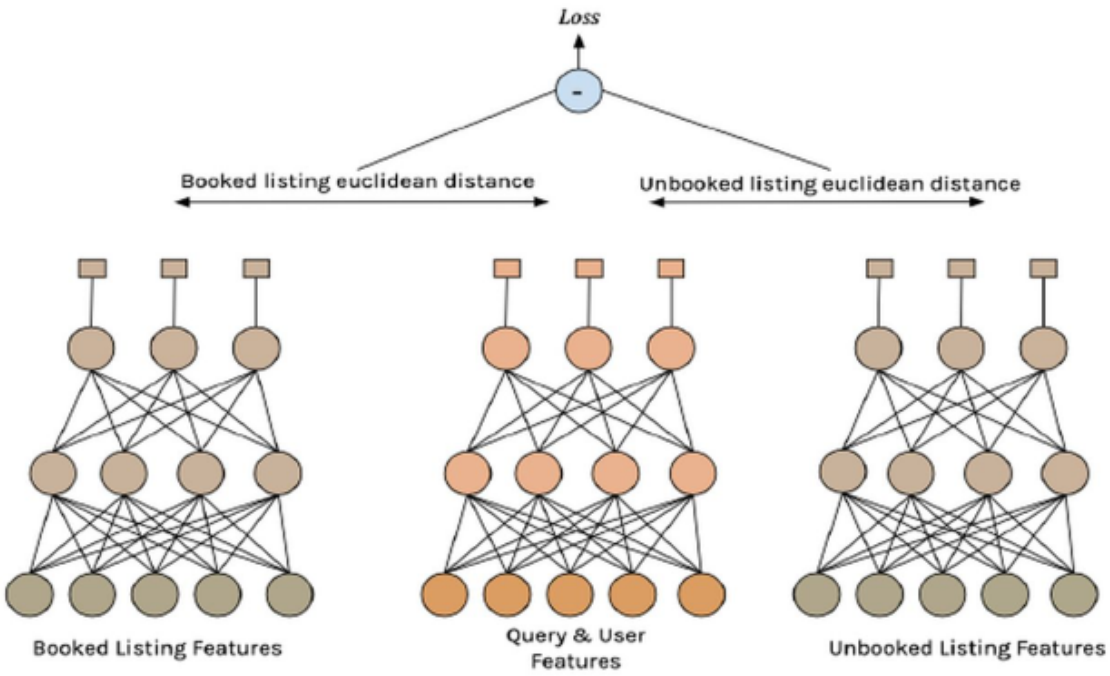


Fig. 6. Pairwise loss computation for training the two tower architecture.

中间的塔是Query&User的特征；左右两侧的塔是<被预订，未预订>的特征。这三个塔分别学习100维的向量，之后计算euclidean distance。

损失函数：基于pair-wise计算，基于Query&User和正样本、负样本的欧式距离，计算差值(Unbooked list euclidean distance - Booked list euclidean distance)，然后和全1向量计算cross entropy loss。这是因为我们希望Booked listing塔的输出更接近Query&User塔；让Unbooked Listing塔的输出更远离Query&User塔。(这也正是对比学习的思想)

模型的简要代码：

```
import tensorflow as tf

def get_tower(features, w0, b0, w1, b1): // 两层 DNN
  """ Two fully connected hidden layers producing a 100-d vector """
  h1 = tf.nn.tanh(tf.matmul(features, w0) + b0)
  h2 = tf.nn.tanh(tf.matmul(h1, w1) + b1)
  return h2

def get_distance_to_ideal(query_vec, listing_vec): // 欧氏距离
  """ Euclidean distance of the listing hidden layer to the query
    hidden layer. In practice minimizing sum of squared diff is
    equivalent to minimizing the Euclidean distance. """
  sqdiff = tf.math.squared_difference(query_vec, listing_vec)
  logits = tf.math.reduce_sum(sqdiff, axis=1)
  return logits

def pairwise_loss(query_features,
                  booked_listing_features,
                  not_booked_listing_features):
  qvec = get_tower(query_features, // 中间塔
      query_w0, query_b0, query_w1, query_b1)
  booked_vec = get_tower(booked_listing_features // 左塔
      listing_w0, listing_b0, listing_w1, listing_b1)
  not_booked_vec = get_tower(listing_features, // 右塔.
      listing_w0, listing_b0, listing_w1, listing_b1)

  booked_distance = get_distance_to_ideal(qvec, booked_vec)
  not_booked_distance = get_distance_to_ideal(
      qvec, not_booked_vec)
  distance_diff = not_booked_distance - booked_distance
  # Push the not booked away and the booked closer to
  # the ideal by increasing relative distance in between.
  xentropy = tf.nn.sigmoid_cross_entropy_with_logits(
    labels=tf.ones_like(logit_diffs),
    logits=logit_diffs)
  loss = tf.reduce_mean(xentropy)
  return loss
```

Table 3. Abstracted TensorFlow™ code for the two tower architecture.

Airbnb这篇文章在排序的时候使用的居然是召回中用的无交互双塔模型，这个在理论上还是很难想通的，毕竟排序的时候我们希望query和item有交互。但是，这个毕竟是在实际应用场景实验多次的结果，而且双塔分离的确可以降低复杂度 -- 对于一个query，我们只需要计算一次query塔的输出结果就可以了，然后去求和query塔点积最大的item即可。

3.【冷启动】问题

冷启动问题是任何推荐系统必须面对的问题，不论是user冷启动还是item冷启动。Airbnb的这篇文章解决的是item冷启动问题。

airbnb关注到冷启动的问题是因为他们发现**新上的item和以前就有的item会有一个6%的NDCG差距。**而且，他们做了一个实验，就是**舍弃掉item所有和用户交互相关的特征**(e.g. the number of past bookings)，模型NDCG下降了4.5%。这说明**DNN很依赖item和user的交互历史信息。**而可惜的是，对于新的item而言他们是没有这部分信息的。

3.1 Approaching Cold Start as Explore-Exploit

解决冷启动问题的一个方法是当作explore-exploit trade-off问题：

- **exploit：利用已知的比较确定的用户的兴趣，然后推荐与之相关的内容**
- **explore: 除了推荐已知的用户感兴趣的内容，还需要不断探索用户其他兴趣**

排序的策略可以通过exploit以往的订单去优化短期下单, 对于长期而言则需要去explore那些新的item。这种权衡其实就是让新item有更多的曝光机会, 从而通过很少的代价收集到用户**对新item的反馈。**airbnb就是通过对新的item进行加权让它有更多的曝光机会（+8.5%）。

但是这种方法带来了一些问题：

- 搜索结果相关性的降低，短期内会降低用户体验。（新item的曝光率太高会导致排序结果相关性的变化）
- 长远来看虽然提升了订单率，但是这种方案缺乏一种明确的目标定义。这样会导致有的人觉得结果好，有的人觉得结果差，不会令人满意。

3.2 Estimating Future User Engagement

为了让系统更加可控，文章回到最开始的问题--是什么导致了冷启动? **其实就是新的item缺少一些用户交互信息**(e.g. number of bookings, clicks, reviews),而其他特征(price, location)是不缺少的。那么，问题变成了：**如何根据已有的特征去预测这些用户交互信息呢？如果我们能够准确地预测出来用户交互信息，那么冷启动问题就解决了！**

baseline方法是用一些default值去填充缺失的用户交互信息；**而Airbnb的方法是用离new item地理位置临近的、且客人数相等的老item的用户交互信息平均值来填充缺失值。**

> For example, to estimate the number of bookings for a new listing with a two person guest capacity, it took the **average** number of bookings for all listings within a small radius of the new listing with a capacity of two.

4. 消除Position Bias

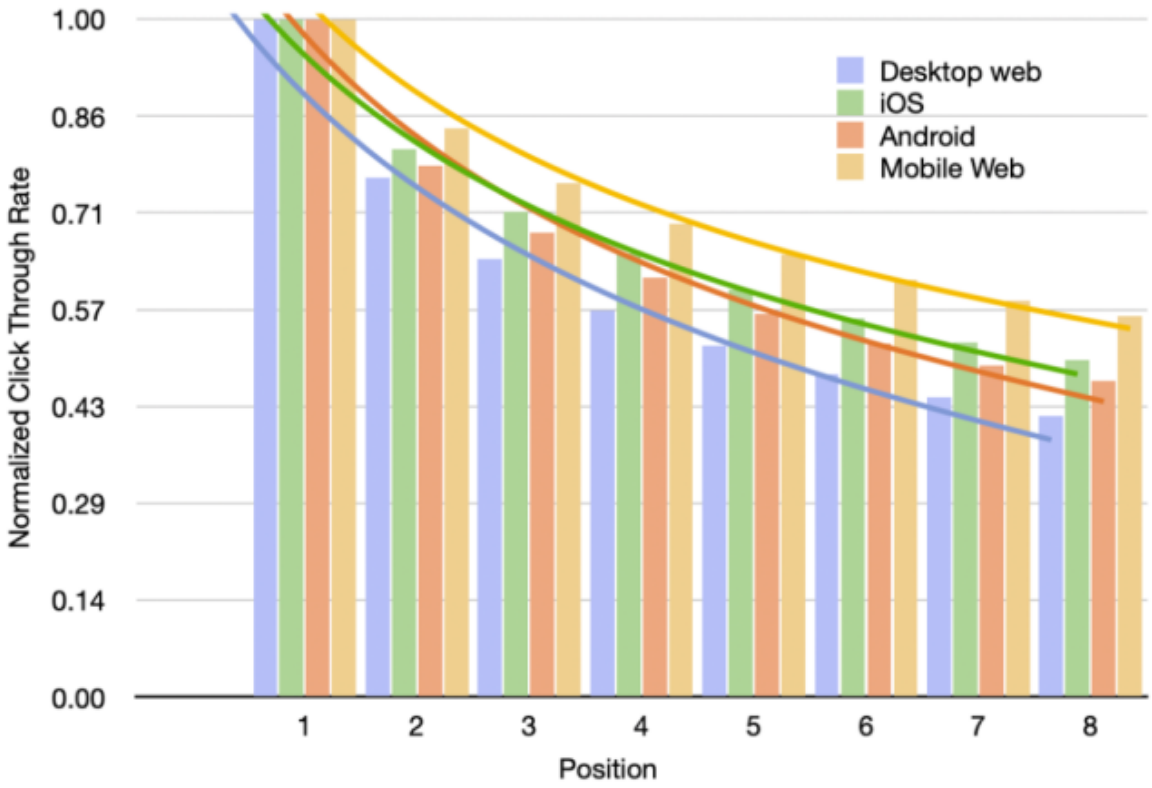


Figure 2. Click through rates by position in search results

给定一个用户 u，以及一个query q 和一个item l，以及list中的每个位置 k。用户预订的概率是：

$$P_{booking} = P(relevant = 1|l, u, q) * P(examined = 1|k, u, q)$$

其中前半部分是这个item被用户预订的概率，后半部分是item在位置k被用户看到的概率。二者相乘就是一个item在位置k上被预订的概率。理想情况下我们只要关注于前半部分然后对list进行相关性排序就OK。

Airbnb在训练时加入位置信息，但是在预估的时候将特征置为0。但是发现模型的NDCG跌了1.3%。文章指出，可能是训练的时候相关性的计算过度依赖位置信息，但是在测试的时候，这个位置信息就没有了，所以导致效果变差。

为了减少相关性计算对position feature 的依赖，文章采用了训练阶段对position feature 进行dropout，这样就能够减少模型对位置特征的依赖。

通过实验文章选择了0.15的dropout比例，对线上的结果有0.7%的**下单率**的提升。经过多次迭代之后，订单收入涨了1.8%。

总结

总的来说这篇文章亮点很多，特别是对深度模型结果的**分析解释**，以及如何通过分析来对症下药。对于现在不可解释的DNN而言，如何**从模型驱动转变为数据驱动**是现在必须面对的问题，特别是后deep learning时代的搜索/推荐。