

# wide&deep learning for recommend systems

原创 改善自我 改善自我 2018-10-20

收录于话题 #DT时代, 终身学习

20个

感觉初次翻译论文，很多不是特别通顺，但是没有经历爬，如何才能会跑呢？坚持学习！！！！

【推荐系统一种广度&深度学习方法】

## 一、概要：

使用非线性特征变换的广义线性模型已经被广泛的应用在大量带有稀疏输入的回归或者分类问题上了。

### **Memorization:**

这里可以理解为特征相互作用的记录，通过对一组特征进行叉积进行特征转换，使得它变得非常的高效，并且解释性很好。然而，如果要泛化可能需要更多的特征工程；

### **Generalization:**

深度学习，可以不怎么做特征工程，相对于稀疏特征可以通过低密度嵌套学得一些隐藏的特征组合。然而，深度神经网络可能会过度泛化，当用户item交互比较稀疏以及高级别时，可能会给永辉推荐不太相关的item。

wide&deep learning 就是通过联合训练线性以及深度模型，将二者的优势进行融合，完成高准确的推荐。

这个推荐算法已经再谷歌play进行了应用，并且线上实验表明用户app购买有了明显的提升，并且谷歌还再tensorflow里面开源了此算法的代码。

## 二、介绍：

一个推荐系统好比一个搜索排序系统，它的输入是用户的一系列的query以及前后上下文信息，输出是一个排好序的items。给一个query，推荐系统的任务就是从数据库中找到相关的items，然后基于一些确定的信息（点击或者购买等）进行rank排序。

推荐系统的一个挑战，和一般的搜索排序问题很类似，那就是同时兼顾Memorization和Generalization；

**Memorization:** 简单来说，就是学习items或者特征的频次以及通过历史数据挖掘可行的相关性；基于此算法的推荐系统在用户已经执行了一些具体行为下对于items显得更加突出以及直相关。

**Generalization:** 主要基于相关性的转换以及探寻新的在原来的数据里面没有或者稍有的发生的特征组合；基于此算法的推荐系统主要为改善多样性的。

这篇论文主要是聚焦谷歌Play商店上的apps推荐，但同时这个也可以应用到一般的推荐系统中。

在工业环境里，对于大量的推荐以及ranking系统，泛化的线性模型比如逻辑回归已经被广泛的使用了，主要因为它们比较简单、可扩展并且可解释。这些模型常常通过one-hot编码成二元稀疏特征进行训练。比如：二元特征『user\_installed\_app=netflix』如果为1，代表用户安装了netflix.这样Memorization就可以通过在稀疏特征进行叉积变换高效地获取到。比如『user\_installed\_app=netflix, impression\_app=pandora』如果值为1，说明这个用户先是安装了netflix，然后播放潘多拉。这也就解释了特征的共现是如何和目标label产生关联的。Generalization是提取更大粒度的特征，比如『user\_installed\_catergory=video,impression\_catergory=music』，但是这个是需要人工进行特征工程的。叉积的一个局限就是泛化不出训练数据中没有出现过的特征对。

像FM或者DNN这些基于embedding的模型，可以通过每个query和特征学习一个低密度的embedding的向量泛化出先前没有见过的特征对，而基本上不需要特征工程。然而，当query-item矩阵比较稀疏and高秩时，很难学习出query和item有效的展现。比如有特殊偏好的用户，或者一个有限面的items，在这些case中，大多数query和item对没有交互，但是对于query-item对密度embeddings将会给出非0的预测，因此这样会过度泛化并且给出不太相关的推荐。然而，通过叉积进行特征变换的线性模型可以以很少的参数记录下这些『异常的规则』。

这篇论文通过 wide&deep learning 一个模型同时获得 Memorization 和 Generalization，同时训练一个线性模型和一个神经网络模型，具体学习框架如图一：

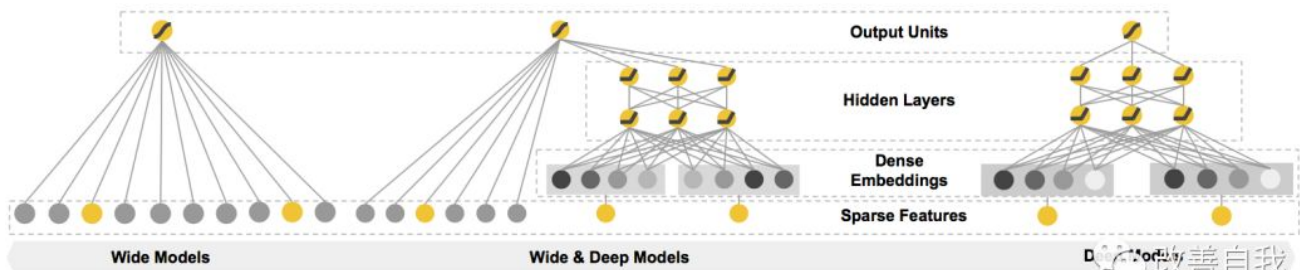


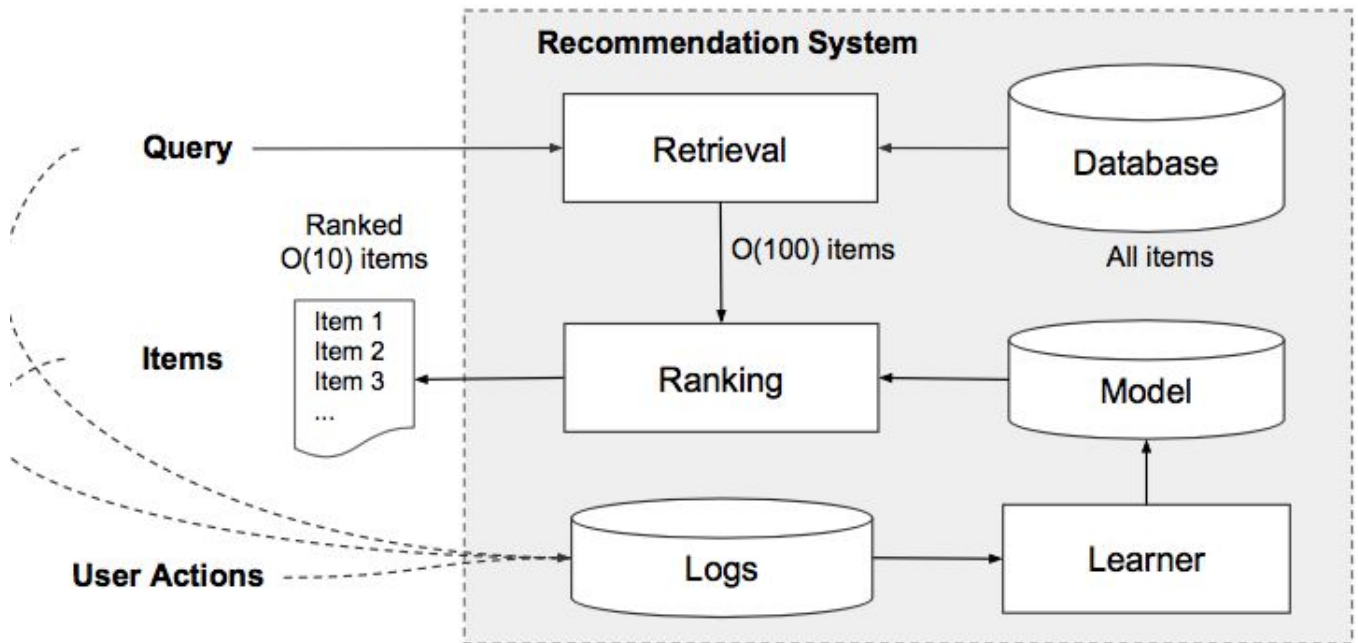
Figure 1: The spectrum of Wide & Deep models.

### 这篇论文的主要贡献：

- 1、一个wide&deep learning的框架，在稀疏的输入一般的推荐系统中，可以联合训练embedding的反馈神经网络和特征变换的线性模型。
- 2、wide&deep 推荐系统的实践和评估已经在谷歌play上进行了产品化，而谷歌play是一个拥有超过10亿用户以及超过1百万app的手机app商店
- 3、此算法的实现已经开源，并且在tenseorflow提供了高质量的API。

推荐系统的全览：

整个推荐系统的架构全览如图二：



**Figure 2: Overview of the recommender system.**

当用户访问app商店时，会生成一个query，包含了各种用户以及上下文的特征。推荐系统会返回一个app的列表，而用户会在这些app上产生行为比如点击或者购买。这些用户的行为，伴随着一系列的query和痕迹，都会记录到日志里面，为后面的学习提供训练数据。

由于在数据库中拥有超过百万的apps，如果对每个query都需要给每个app进行打分，并且服务在ms级给出响应，这个是非常难处理的。因此，我们收到一个query后的第一步就是检索，这个检索系统会通过使用各种型号返回一个短的匹配的比较好的item列表，这些信号一般都是基于机器学习的模型和人工定义的规则的联合。通过减少候选池，推荐系统通过items的打分进行排序。这些打分一般都是 $P(y|x)$ ，基于x特征下发生用户y标签行为的概率，x特征包含了用户特征（城市、语言等），上下文语意特征（设备、时间）、以及日志特征（app年龄，历史统计数据）。这篇论文主要使用wide&deep learning 架构的推荐模型上。

### 三、wide&deep 学习：

#### wide部分：

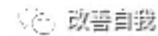
wide 部分就是一个广义的线性模型，模式就是 $y=w*x+b$ ；y是预测值， $x=[x_1, x_2, \dots, x_d]$ 是一个向量，d个特征； $w=[w_1, w_2, \dots, w_d]$ 模型特征的系数，b是偏置项。特征向量包含了原始特征和转换后的特征，一个很重要的变化就是叉积转换，定义如下：

$fik(x) = x_i^{cki}$  cki属于{0,1}；cki是一个bool值，1代表如果i个特征是第k个转换的一部分，0标示不是；对于二元特征，一个叉积转换：AND (gender=female, language=en)，如果是1，只有当gender=female 和 language=en都为1，否则为0；这样就呈现了特征的交叉，为广义的线性模型增加了非线性性。

**deep部分:**

前向反馈神经网络。对于分类的特征，原始的输入是特征字符串。每一个稀疏，高维类别的特征会首先转换为低维且密度的实数向量，经常成为embedding向量。embedding的纬度经常是按照从低到高的顺序。在模型训练中，embedding向量开始随机然后经过训练最终最小化loss函数。在前向神经网络中，这些低纬度密集embedding向量会流入到隐藏层。特别是：每一个隐藏层的计算如下：

$$a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)})$$



$l$ 是层次， $f$ 是激励函数； $a$  $b$  $w$ 分别是激励、偏差以及权重（ $l$ 层的）；

**联合训练wide&deep模型:**

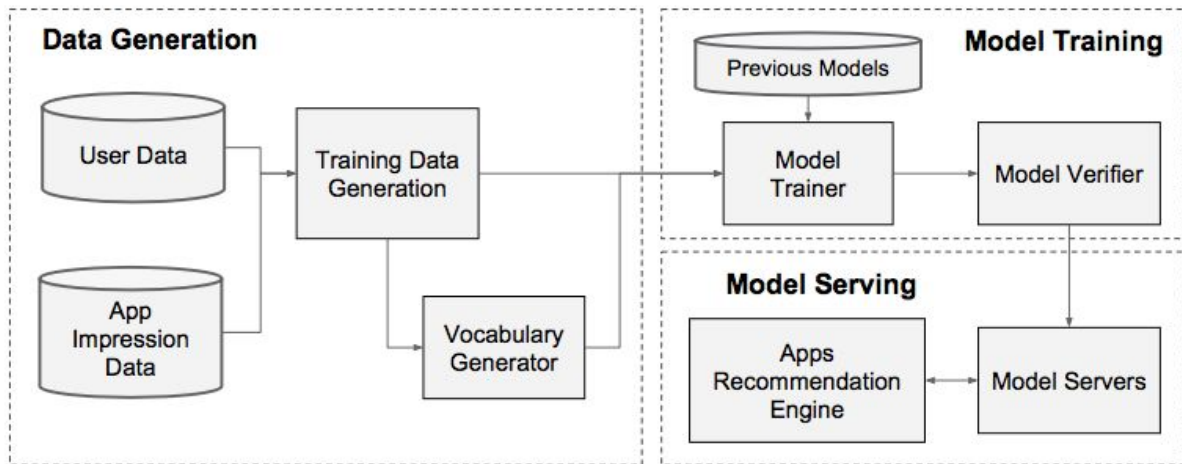
wide部分和deep部分最终使用一个加权求和取log的方式进行预测。这里要区分一下joint learning和ensemble的区别：ensemble：独立的模型都是分开训练的，相互没有关系，只有在预测的时候才会合在一起。相反，joint learning：是在训练的时候将wide和deep的部分以及权重和放在一起进行优化所有的参数；在模型大小上也有区分：ensemble由于是分开训练的，更多的特征和变换，去获得合适的准确率；所以模型大小会比较大。相反，相比全size的wide模型，joint learning wide的部分只需要一个很小部分的叉积弥补deep part的弱势就好了。joint training wide&deep模型是通过反向传播对wide和deep部分进行梯度迭代，小样本随机优化的。在实验中，我们使用了FTRL算法对于wide部分的模型，adagrad对于深度的部分。

预测模型公式如下：

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$


**四、系统实现**





**Figure 3: Apps recommendation pipeline overview.**

where  $l$  is the layer number and  $f$  is the activation function, often rectified linear units (ReLUs).  $a^{(l)}$ ,  $b^{(l)}$ , and  $W^{(l)}$  are the activations, bias, and model weights at  $l$ -th layer. 改善自我

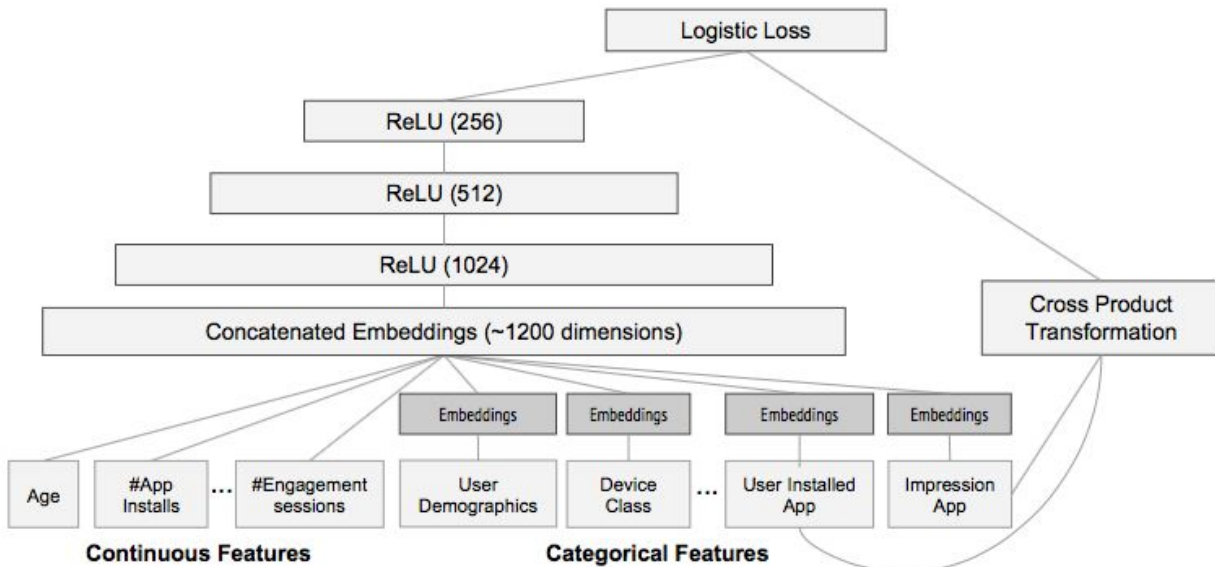
### 数据生产：

在这个阶段，用户和app的一段时间的使用数据会用来生成训练数据。每一个样本都会对应一次使用，label就是用户的获取（1-代表安装，0-代表未安装）。

类别特征和ID的映射也是在这一步产生的，系统会对每一个字符串特征计算它的ID空间。连续的实数特征值会被归一化到【0，1】。

### 模型训练：

模型训练架构如图四：



**Figure 4: Wide & Deep model structure for the apps recommender.**

改善自我

输入层输入训练数据以及词汇且生成带有label的稀疏以及密度的特征。wide部分又积转换用户安装以及使用的apps。对于deep部分，对于每一个类别特征都会学出一个32维的embedding向量。我们将所有特征放在一起将近1200维，这些特征最后喂到3层的神经网络中，最后逻辑回归输出。

wide&deep模型训练5000亿的样本，每次新的训练数据过来，模型都需要重新训练，然后重新训练成本非常搞，耗费很多时间。为了解决这个问题，对模型做了一个温启动操作，每次初始化为上次的模型。在加载前一次的模型前，需要进行检查，没问题才可以进行加载。

#### 模型服务：

一旦模型被训练好并且被检查通过，我们就会将模型加载到模型服务上去。然后server会根据召回系统的返回结果进行排序打分，最后给出各个app顺序；

为了能够使服务10ms级出结果，这里使用了多线程并行技术，多线程并行计算各个候选app的打分，而不是在同一个线程中计算。

#### 五、实验结果：

##### app收益

通过abtest结果显示，收益明显

##### 服务表现

使用多线程技术大大提升了效率；

后面省略。。。